



Escola Tècnica Superior d'Enginyeries  
Industrial i Aeronàutica de Terrassa

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Degree Thesis

# G-Engine: Microcontroller based video game console



Grau en enginyeria electrònica industrial i  
automàtica

Author: Guillem Marcet

Tutor: Gabriel José Capellà

June 2018



## Annexes

Annex 1.1 differences between product lines in the F1 family. ....	58
Annex 1.2 Differences between configurations in the F103 line. ....	58
Annex 1.3 Development board used to develop the prototype. ....	58
Annex 1.4 LCD front view. ....	59
Annex 1.5 LCD back view. ....	59
Annex 1.6 Micro SD card to SPI adapter ....	59
Annex 1.7 Battery charging PCB. ....	59
Annex 1.8 Battery 3.7V 5200mAh ....	60
Annex 2.1 ST link v2. ....	60
Annex 3.1 Memory map. ....	61
Annex 3.2 Boot loader USART instructions. ....	62
Annex 3.3 TIM2_CR1 register. ....	63
Annex 3.4 ADC_SR. ....	63
Annex 3.5 ADC_CR2. ....	64
Annex 3.6 ADC_SQR1 ....	64
Annex 3.7 ADC_SQR3 ....	65
Annex 3.8 ADC_DR ....	65
Annex 3.9 SPI registers. ....	66
Annex 4.1 HX8357-b instruction list 1. ....	67
Annex 4.2 HX8357-b instruction list 2. ....	68
Annex 4.3 HX8357-b instruction list 3. ....	69
Annex 4.4 HX8357-b instruction list 4. ....	70
Annex 5.1 SD card commands 1. ....	71
Annex 5.2 SD card commands 2. ....	72
Annex 5.3 Application SD card commands ....	73
Annex 5.4 CSD structure. ....	74
Annex 5.5 BPB in FAT32 ....	75
Annex 5.6 directory Entry byte structure. ....	78
Annex 5.7 Long Name directory entry byte structure ....	79

Annex 6.1 Main: .....	80
Annex 6.2 Fat32.h and FAT32.c.....	82
Annex 6.3 SD Card .....	88
Annex 6.4 Fat directory- lcd interface.....	93
Annex 6.5 controller input gathering .....	96
Annex 6.6 Game code .....	100
Annex 6.7 Timer 2 configuration.....	124
Annex 6.8 ADC configurtation .....	125
Annex 6.9 Timer 2 interrupt.....	128
Annex 6.10 LCD GUI .....	130
Annex 6.11 LCD.h .....	133
Annex 6.12 LCD.c.....	138
Annex 6.13 SPI.c .....	143
Annex 6.14 Sprite Arrays.....	145

## 0. Abstract

The objective of this project is to develop a functional portable videogame console for playing retro/arcade games that is capable of loading games from an SD card. For that, the device will be build around a stm32 microcontroller, specifically the stm32f103ZE.

To accomplish this objective, this document will detail the operational processes that occur in the microcontroller, how to configure these processes, how to communicate with the development board and with other modules and how to program it. The same goes for the LCDs microcontroller and for the SD card interfaces and FAT32 file system. All this implies a good knowledge of the programming language C and the ability to find and understand reliable documentation.

The console, named G-Engine, will be a portable device for playing arcade games. It will display the game on a 480x320 pixel resolution LCD, and the user will control it using its 8 input buttons: four directions buttons, two action buttons and the select and start buttons.

G-Engine will be able to load games from an SD card making it so the user can have multiple games installed on the card and can change from one to another any time. It will implement the FAT32 file system so the games can be installed on the SD card from any PC by just “click and drag” the game into the card.

To ensure portability the device has a 5200mAh Li-Ion battery that can be charged through a mini-USB connector. It will also have three battery status LEDs. One indicates low battery, other indicates that the device is being charged and the last indicates that the charge has finished. In order to reduce the power consumption the device has a power switch to disconnect the battery from the processor when the gaming session is over.

## 1. Component description

### The microprocessor

The chosen microcontroller was the STM32 F103 ZE. A project of these characteristics needed a powerful and fast enough processor and the STM32 line offers chips with a 32-bits architecture at a very reasonable price. Of all the processors in this line the F103 was readily available in different online markets and had a lot of General purpose In/Out pins (112 to be specific).

In this project there was a need for a lot of I/O, just the LCD uses around 25, the controller board with the input buttons occupies 8 more, and the SPI bus, the battery management and the LED indicators add up about 10 more. 112 I/O pins may seem an exaggeration but the differences in price were insignificant and it was preferable to be safe rather than sorry. This also leaves a lot of room for improving the hardware in the future. Finally, the ZE configuration offered a really good amount of flash memory (512kBytes) removing the need to worry if the final program would fit or not in the microcontroller memory, this is important in game programming because the graphics and sounds take up a lot of memory space.

More information about the STM32 families can be found in annexes 1.1 to 1.3

The processors itself must be powered with a voltage range of 2-3,6 volts, but the board it's mounted on has an internal regulator that can allow voltages up to 6V. This board also contains all the components the processor needs to work (like quartz resonators for the oscillator, resistors, and capacitors), a mini USB connector for general purposes and a J-TAG for programming the chip.

This chip has a lot of peripherals, USART, SPI, ADC, DAC, Ethernet... but in this only ones that will be used are:

- USART, for programming. Though it's handled by keil and st-link all by themselves
- SPI. To retrieve information from the SD card.
- ADC. To read the battery level and activate the alert LED.
- Timers. They will help to program temporal processes.

### The LCD

The display used in this project was a 480x320 colour pixel LCD with a size of 3,5" and driven by the microcontroller HX8357B. Each pixel has a 16-bit colour resolution, or, in other words 65.536 different colours.

Although the LCD board has 36 pins, some of them are NC (not connected). The useful pins are:

- 4 pins for power supplying
- 16 pins for data and commands
- 4 pins for data control.

It also has a SD card slot and 4 pins that are dedicated to communication with it, but the location of the adapter makes it impossible to access the SD card once the device is assembled and will not be used in the project.

Images of the LCD module used can be found in annexes 1.4 and 1.5.

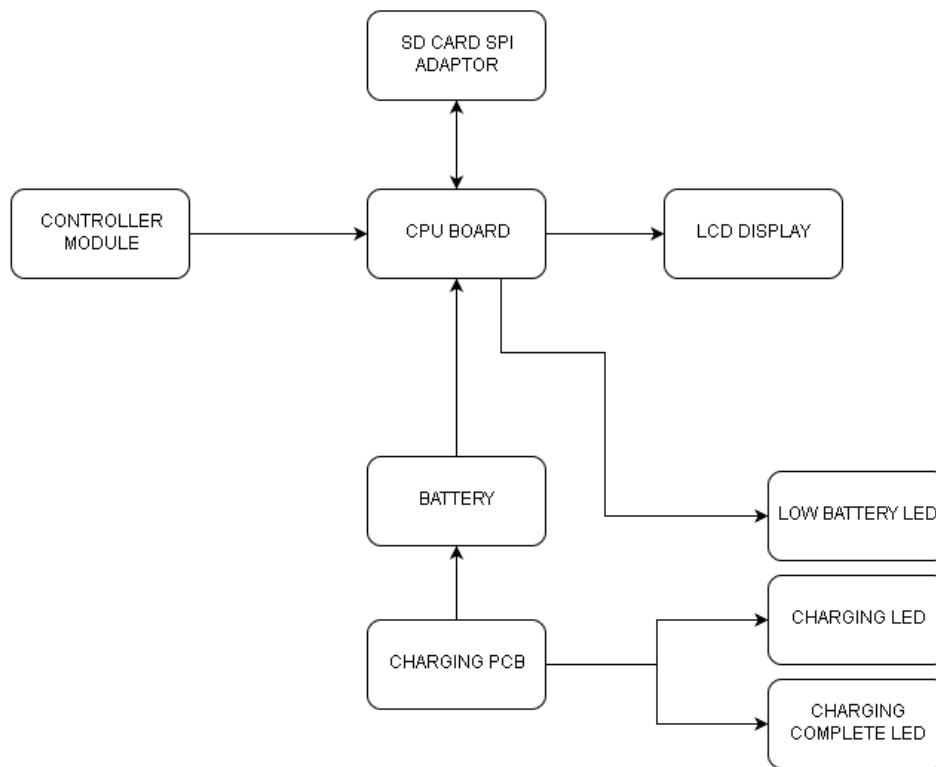
## Other components

Besides the above, in this project were also used:

- 8 Push buttons that conform the controller module.
- A two position switch for ON-OFF the device.
- A Micro SD card to SPI adaptor: this small board makes the SPI pins of the SD card available. The PCB contains the following components:
  - o A voltage regulator, in case the supply voltage is different than 3,3V.
  - o A buffer IC to smooth SPI communication and convert the high logic level from 5v to 3v3 in case of need.
  - o A slot for a micro SD card.
  - o I/O pins for SPI communication and power.
- Battery charging PCB based on TP4056: controls the charging cycle of the battery, has LED indicators for “charging” and “charge complete” states. The power input is made through a mini USB adapter.
- 3.7V Battery rescued from a broken tablet with a capacity of 5200mAh.

Several component images can be found in annexes 1.6 to 1.8.

Image 1.1 shows the connection diagram between components.



*Image 1.1 Component connection diagram*



## 2. Tools

### Hardware Tools

- ST-Link v2 to program and debug the controller. This adapter connects itself to the PC using a USB connector and connects to the development board with the J-Tag connector. An image can be found in the annex 2.1.
- Breadboard
- Connection cables
- Cables for welding
- Soldering iron
- LEDs

### Software Tools

- µVision 5: code development software
- STM32F10x libraries. They contain all the methods needed to easily program and configure the microcontroller
- HX8357 libraries. They contain all the methods needed to configure the LCD and show images on it.
- Solid Edge ST5 for designing the case.

### 3. Processor configuration

#### Memory Map

Having a 32-bit address bus, the stm32F103ZE processor can address up to 4 GB. Nevertheless, the vast majority of these positions are either not linked to real memory or peripherals. An important share corresponds to the flash memory (which is where the program will be) and the RAM (remember that this processor has 512KB of flash memory and 64KB of RAM).

The rest is distributed among the configuration registers, which dictate how the different peripherals will perform, and memory and peripherals used only by the processor.

In the reference manual provided by the supplier we can see that the memory is divided in 512MB blocks. Each block has addresses that have related functionality.

- Block 0: Code (0x0000 0000 – 0x1FFF FFFF).  
It corresponds to the code memory space.

The first 512KB are aliased to either Flash or system memory depending on the boot pins. This means that depending on the boot mode of the processor, a call to an address in this range will actually call a memory in the flash region or the system memory region.

A RAM access can also be casted from this region. This is explained in the next Block.

Starting at 0x0080 0000 and continuing for 512KB the Flash memory can be found. The code will be here.

Starting at 0x1FFF F000 and continuing for 2KB the system memory can be found. Here is the program that allows writing code on the flash memory through USART.

Immediately after system memory region 16 Bytes corresponding to Option bytes can be found. These configure the basic behaviour of the processor and they don't change after a system reset. Option Bytes configure things like: from which flash memory address starts the execution or allow enable/disable read/write protection, among others.

The rest of the addresses are reserved, that means that they are either not-linked to anything or that the processor uses them for its internal processes.

- Block 1: RAM (0x2000 0000 – 0x3FFF FFFF).

This processor has 64 Kbytes of RAM with single bit access. To be able to access a single bit, the processor uses the addresses of the aliased region in Block 0. In 64Kbytes there are 512Kbits and that is the exact number of addresses in the aliased region of block 0, each one redirecting to a bit in the RAM region in Block 1.

The rest of this block's addresses are protected.

- Block 2: Peripherals.

In this block we find the registers that configure the peripheral options like: the baud rate of serial communications, turn to high a GPIO pin or configure the reference voltage of an ADC. This will be seen in detail on its corresponding section.

- Block 3 and 4: FSMC banks:

The STM32F103 has the capability of adding extra flash or ram memory via external chips using the FMSC (Flexible Static Memory Controller). But in this project there is no need to use this capability.

- Block 5: FSMC registers  
This contains the registers to use and configure the FSMC
- Block 6: Is Reserved
- Block 7: Is used for the processor internal peripherals

A detailed memory map found in the stm32f10x manual can be found in annex 3.1.

### Programming the chip:

The processor can fetch the instructions from 3 different memory regions. It has two pins to determine which one of them is actually used, depending on the state of these pins (high or low) the boot region is selected (table 3.1).

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as boot space
0	1	System memory	System memory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM is selected as boot space

*Table 3.1*

To load the code in the flash memory the processor needs the system memory to be selected as boot space. This means that the instructions come from the System Memory region which, as seen in the memory mapping, has the boot loader code.

The other boot spaces are: Flash memory (that is where the code will go) and Embedded SRAM.

When running the boot loader, the processor is configured to receive instructions via USART through the corresponding pins. There are plenty of instructions that the processor understands (they can be found in annex 3.2) but the more important is the “write memory” command (0x31). This instruction passes the processor a starting address and the data to be written. That data will be the code of this project and will execute when flash memory boot mode is selected.

The USART baud rate needs to be between 1.200 and 115.200 and the frame structure is: start bit, 8 data bits, even parity bit and one stop bit.

Luckily, the chip programming is handled by the st-link hardware in conjunction with keil uvisio. Also, the selected development board has an USB to serial chip so it is possible to send the instructions through the USB port of a PC using a simple application that sends and receive data through the USB to a serial adapter.

## Configuring the I/O

The processor has 112 input/output pins. This section will talk about how to configure the input and output and also how to use the alternate function (USART, SPI...) they have.

This 112 pins are grouped in 7 ports (A, B, C, D, E, F, G) with 16 pins each (0-15). Each of the general-purpose I/O ports has the following registers (the x stands for the port letter A...G):

- Two configuration registers (GPIOx\_CRL and GPIOx\_CRH)
- Two data registers (GPIOx\_IDR and GPIOx\_ODR)
- One set/reset register (GPIOx\_BSRR)
- One reset register (GPIOx\_BRR)
- One locking register (GPIOx\_LCKR).

Note that all registers are 32 bit long

### Configuration Registers (GPIOx\_CRL and GPIOx\_CRH)

The configuration register allow individual pin configuration. The CRL (configuration register low) configures pins 0 to 7 and the CRH (configuration register high) configures the pins 8-15. Each pin needs two fields of two bits each to be configured, they are MODE and CNF. The GPIOx\_CRL structure is found in image 3.1 (the same structure goes for GPIOx\_CRH but for pins 8-15):

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

*Image 3.1 GPIOx\_CRL structure*

The mode field configures whether the pin will be an input or an output pin, and in case it's an output pin, it also configures its switching frequency (see table 3.2 for details).

MODE[1:0]	Meaning
00	INPUT
01	OUTPUT, 10MHz
10	OUTPUT, 2 MHz
11	OUTPUT, 50MHz

*Table 3. 2*

The CNF field means different things for input or output mode.

For input the pin can be set pin to do analog or digital readings. In the case of digital readings one can set a pull-up, pull down o leave it as floating input (See table 3.3 for more information).

If an input pin is left floating it will be more susceptible to false readings due to ambient noise. The pull input solves this using a resistor between the input pin and the default value, this default value is digital high for pull up, or ground for pull down.

To select between the pull up and pull down, the register ODR is used. Writing this register sets the output value if in output mode or the default value if in input mode.

CNF[1:0]	Meaning
00	ANALOG
01	FLOATING INPUT
10	PULL UP/PULL DOWN INPUT
11	Reserved

*Table 3. 3 CNF in input mode (MODE=00)*

For the output the CNF[0] makes the pin to behave as push-pull or open drain. And The CNF[1] selects whether the pin will work as a general purpose pin or will work as its alternate function(SPI,USART...). (See table 3.4 for more information)

CNF[1:0]	Meaning
00	General purpose push pull
01	General purpose open drain
10	Alternate function output Push-pull
11	Alternate function output floating

*Table 3. 4 CNF in output mode (MODE>00)*

This register reset value puts all pins in MODE: input and CNF: floating input

### **Data registers (GPIOx IDR and GPIOx ODR)**

These registers are differentiated between Input Data Register (IDR) and Output Data Register (ODR).

The IDR is read only; it can't be modified by software by any means. It contains the input value the corresponding IO pin. (See image 3.2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

*Image 3. 2 IDR structure*

The ODR on the other hand can be read and write. And it sets the output value on the corresponding IO pin. The structure of this register is the similar to IDR's. (See image 3.2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

*Image 3.3 ODR structure*

The only “problem” here is that the only way to access this register is as a full word. This means that is not possible to change one single bit, it's necessary to rewrite the whole register. This is solved using the next register.

### **Set/reset register (GPIOx BSRR)**

Using this register one can bypass the need for rewriting the whole register in order to change the output value of one IO pin.

Setting any of the low 16-bits of this register, the corresponding bit in the ODR gets set. Contrariwise setting one of the high 16-bits the corresponding bit in ODR gets reset.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

*Image 3.4 RSRR Structure*

### **Reset register (GPIOx BRR)**

This is redundant with the high 16-bits of the BSRR. It has the same function and working process but using the lower 16-bits of the register instead of the higher ones. It's useful for specific cases of code optimizing.

**Locking register (GPIOx\_LCKR)**

This register is used for locking the configuration of the corresponding pin until next reset. We will not use this register in our project



## The controller

This console is build to have 8 input controls in order to be like other classic game consoles (see image 3.5).These inputs are read using port E. Check Table 3.5 for more details.

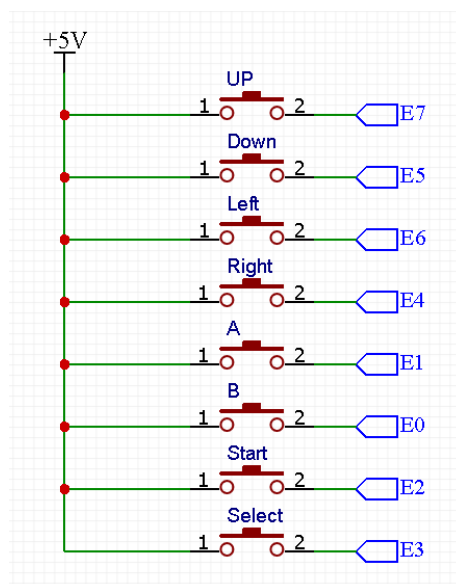


*Image 3. 5 Game boy and NES controller(from left to right). Inspiration for G-Engine controller.*

Control	Port E pin
Up	7
Left	6
Down	5
Right	4
Select	3
Start	2
Action A	1
Action B	0

*Table 3. 5 G-Engine controller pinout*

These Pins are all configured as input pull-down and after pressing the corresponding button the pin is driven HIGH. The controller module schematic can be found in Image 3.6



*Image 3. 6*

To know what button (or buttons) are pressed the GPIOE\_IDR must be read. More details will be found in the programming section.

## Configuring TIM2

In almost any game it is important to control time in one way or another, things that happen periodically, timed events, countdowns....

Normally this time control is made using timers, a peripheral that can count clock pulses. Knowing the frequency of the clock and the count value is easy to calculate the time passed between two readings.

The F103 processor has 14 timers, but in the example game there is only need for one. This section will approach on how to configure the timer used and the code section will focus on how it was used.

Out of the 14 timers, Tim1 and Tim8 have specific functions tied to other peripherals so it's better not to use those. The remainder timers are general purpose timers so there is no problem in using them. In this project Tim2 will be used but any other timer would work as well.

Before using a timer it's important to know how it is configured. This configuration includes:

- Mode of operation: the timer can be configured to count up or count down.
- Value of the prescaler: the prescaler can divide the counter clock frequency by any factor between 1 and 65536. If prescaler=1 the frequency of the timer clock is the same as the processor clock (72MHz).
- Auto-reload value (ARV). In count mode up, the counter value goes from 0 to the auto-reload value and then restarts from 0. In countdown mode the counter value goes from the auto-reload value to 0 and restarts from the auto-reload value.

Timers in this processor have several registers but a lot of them are tied to specific functions with other peripherals like DMA interrupts or generating PWM signals. In this project the only registers that need to be configured or read are:

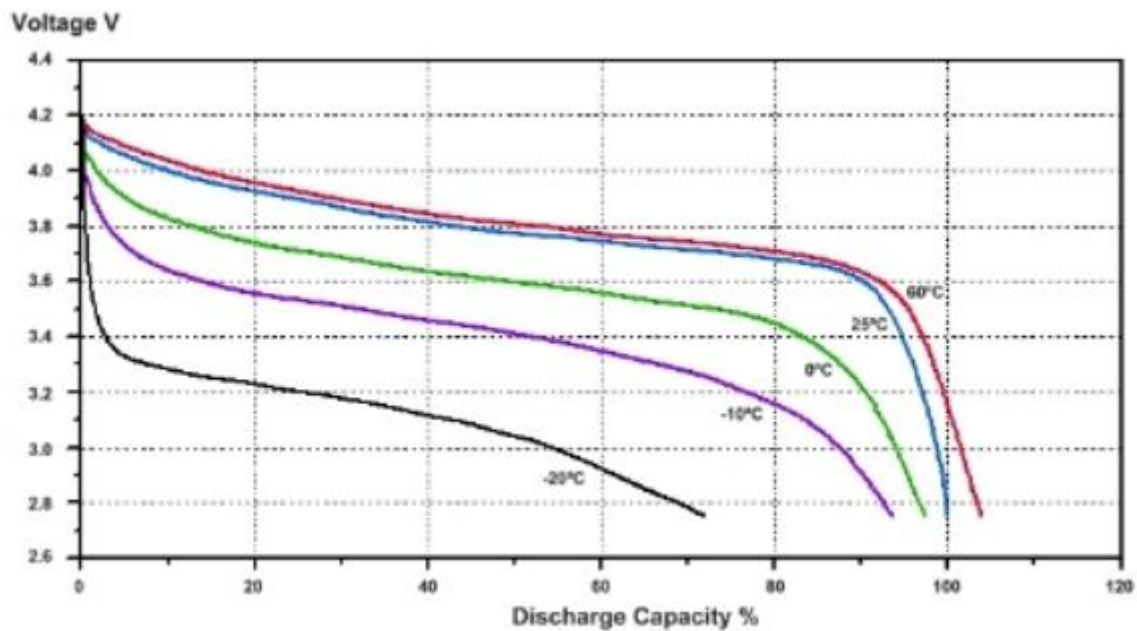
- Counter register (TIM2\_CNT) here is where the counter value is stored, can have values from 0 to the auto-reload value.
- Prescaler register (TIM2\_PSC) used to configure the prescaler, can have values from 0 to 0xFFFF.
- Autoreload register (TIM2\_ARR) used to configure the autoreload value, can have values from 0 to 0xFFFF.
- Control Register (TIM2\_CR1). Specifically the DIR field in CR1 that sets the counting mode up(0) or down(1). This register also configures if the timer is going to restart after getting to the ARV or if it stops or if the timer is going to use the ARV at all. (see annex 3.3 for details on the CR1).

In this project, TIM2's CR1 was left to reset value which means up counting, using the ARV and resetting after getting to ARV. The prescaler was set to 0x8CA0 (36.000 in decimal) and the auto reload value of 0xFFFF (65535 the maximum). This gave a time resolution of 500 microseconds ( $72\text{MHz}/36000=0.0005\text{ s}$ ) and a time between 0 and 0xFFFF in the counter of about 32 seconds. How this counter is used will be explained in the code section.

### Reading the battery level

Batteries don't always output the same voltage. As battery capacity runs out output voltage goes down. This can be used in conjunction with one of the processors analog digital converters to know the battery level, and in case this level is too low turn on the battery low LED attached to pin G0.

The battery used in this project is a lithium-ion battery with a nominal voltage of 3.7V and a capacity of 5200mAh. In image 3.7 we find the discharge curve for li-ion batteries, this can be used to set a threshold, any voltage under that threshold will be considered as low battery.



*Image 3. 7 discharge curve. Note that it's temperature dependant*

In this project the threshold was set at 3.6V because, as seen in the discharge graph (image 3.7), this value allows some time of reaction in the worst case (the battery will still have at least 5% capacity even at 60°C)

The processor has 3 ADC modules with 17 channels each. In this project only one channel of one ADC will be used so to choose a channel and ADC module only the pin availability was taken in consideration. In this project the chosen pin for the ADC function was pin B0 that links with channel 8 in ADC1.

To enable the ADC1 module, the bit ADON in the ADC1\_CR2 register must be set.

Each module allow establishing a sequence of channel conversions; it is possible to make a program that using only one ADC module make conversions in different channels in a particular order. In this project this will not be necessary so the appropriate configuration is the following:

- Set the length of the conversion sequence to 1. This is made writing 0x01 in the L field of ADC\_SQR1 register (Annex 3.6).
- Select the appropriate channel to be the first in the sequence. This is made by writing the channel number in the SQ1 field of the ADC\_SQR3 register (Annex 3.7). In this case this field's value must be 0x08.

To trigger a conversion ADON bit in the ADC1\_CR2 (Annex 3.5) register needs to be set again.

The conversion is not instantaneous; when it has finished successfully the EOC flag (end of conversion) in the ADC1\_SR register is set (Annex 3.4). After that the converted data will be in the ADC1\_DR register's 12 LSB bits. (Annex 3.8)

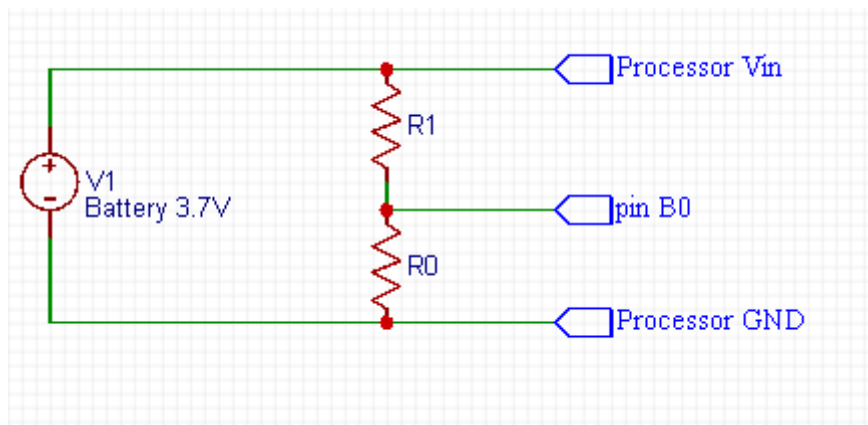
The ADC reference voltage must be a value between 2,4V and 3,6V and the easiest thing to do is to connect the reference voltage pin to one of the 3,3V pins in the development board. This limit is below the threshold decided for the low battery alert. To make use of this ADC it will be necessary to process the battery voltage before connecting it to the ADC pin.

The easiest treatment is a voltage divider. With a pair of high enough and equal resistances is easy to halve the battery voltage signal before connecting it to the ADC. This way the threshold that was decided before needs to be halved as well. The new threshold will be 1,8V.

To convert the value in the ADC1\_DR register is as simple as applying the following formula:

$$V = 3.3 * \frac{ADC1\_DR}{4096}$$

Finally if this value is lower than 1,8V, the G0 pin need to be set to HIGH turning on the red LED.



*Image 3. 8 Voltage divisor schematic. R0=R1=10KOhm*

## Using the SPI

The SPI protocol will be used to communicate with the SD card. In this section all the information regarding the programming of the SPI in the processor can be found. How the actual communication with the SD card works will be detailed in the corresponding section.

This processor has three SPI modules. It doesn't matter which one is chosen because all three work the same way. The decision will be made entirely based on the pins that they use.

The chosen module was the SPI2 since its pins were not used by any other peripheral and they were accessible. Pins used by SP2 can be found in table 3.6

Before going into detail about how to program the processor to implement SPI communication let's see how the communication itself works.

In SPI, which stands for serial peripheral interface, there can be one master and many slaves (in this project there will only be one slave, the SD card). Only four pins are necessary (not including power).

- CS. Chip select, enables the appropriate slave to send information to. A disabled slave doesn't read data from the data bus.
- Sclk. Signal clock, necessary to synchronize the data communication. Is generated by the master
- MISO. Master in Slave out. Through this line the slave will send serial data to the master.
- MOSI. Master out slave in. Through this line the master will send serial data to the slave.

SPI function	Processor pin
NSS	B12
SCK	B13
MISO	B14
MOSI	B15

*Table 3.6 SPI2 pin out*

Master and slave need to agree in the data frame structure.

- Length of the data frame. Either 8 or 16 bits
- MSB first or LSB first
- CPOL and CPHA of the signal.
  - o CPOL. Clock polarity, CPOL=0 the leading edge is clock's rising edge, if not leading edge is falling edge.
  - o CPHA. Clock phase, CPHA=0 output data goes out on clock's leading edge and input data goes in trailing edge, if not it goes the other way around.

As to configure the appropriate registers for a correct SPI communication:

Almost all configuration is done in SPI2\_CR1 register:

- Select the baud rate of the communication using the BR field in. Possible values are found in image 3.9
- Select the CPOL and CPHA bits according to slave requirements.
- Select the DFF bit to implement 8-bits frames or 16-bits frames
- Configure the LSBFIRST bit to match frame format
- Enable the NSS hardware mode. This will make the NSS pin to select the slave while transmitting data automatically.

In SPI2\_CR1 register

- SSM (Slave Select Mode) bit needs to be reset (SSM=0, for hardware mode) and the SSOE (Slave Select Output Enable) bit needs to be set (SSOE=1, to enable output)
- Full duplex communication is needed. This means the sending and receiving of data can occur simultaneously. This is enabled with the bits BIMODE and BIDIOE reset to 0.
- The MSTR and SPE bits must be set to ensure the processor acts as a master and SPI is enabled

All the mentioned registers can be found in the annex 3.4.

Note the following information about SPI transmissions

- The transmit sequence starts when a value is written in SPI2\_DR register. TXE flag is set when transmission starts and the next frame can be written in the buffer without problems. TXEIE in SPI2\_CR1 enables the interrupt for that flag.
- Receive sequence. Data transmitted is found reading SPI2\_DR when RXNE flag is set. Interrupt is generated if RXNEIE bit is set in SPI2\_CR1.
- A read of SPI2\_DR register addresses the RX buffer and clears RXNE flag while a write addresses the TX buffer and clears the TXE flag. This way the same register can handle both data packages without collision.

#### Bits 5:3 **BR[2:0]: Baud rate control**

000:	$f_{PCLK}/2$
001:	$f_{PCLK}/4$
010:	$f_{PCLK}/8$
011:	$f_{PCLK}/16$
100:	$f_{PCLK}/32$
101:	$f_{PCLK}/64$
110:	$f_{PCLK}/128$
111:	$f_{PCLK}/256$

*Image 3. 9 Baud Rate selector.  $f_{PCLK}=72\text{MHz}$*

For communication with the SD card the values of the previous setup need to be the following:

- CPOL = CPHA = 0
- Baud rate between 100 and 400 KHz -> BR=111 -> 281.25KHz
- DFF =0 to select 8 bit frame format
- The transmit needs to be MSB first so the bit in register CR1 LSBFIRST=0

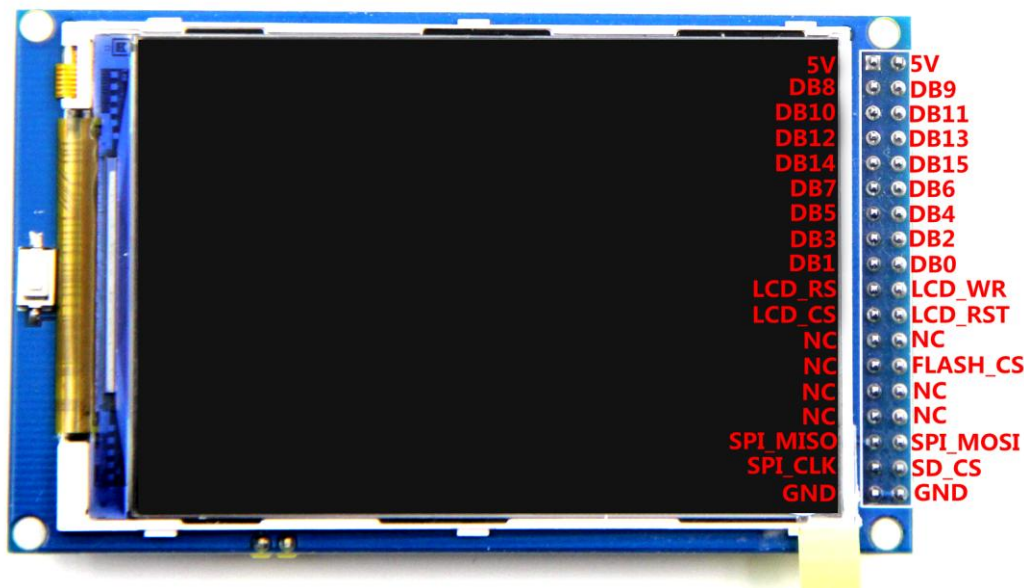
## 4. Configuring the LCD

### Talking to the LCD

As mentioned, the LCD screen used is controlled by the HX8357-b microcontroller. This section will focus on learning how that microcontroller works.

The LCD board has 36 pins (can be found in Image 4.1):

- 4 power pins (2xVCC and 2xGND)
- 16 data/command pins (DB0...DB15)
- 4 control pins:
  - o LCD\_RS the data/command select pin. If held low, means the content of DB0...DB15 is a command code, otherwise means it is transmitted data.
  - o LCD\_WR writepin. The chip reads the DB pins after a rising edge on this pin.
  - o LCD\_CS chip select. If it's not 0, interact with any of the board pins is not possible.
  - o LCD\_RST if held low for at least 100ms, the chip is reset. All configuration and memory values are set to default value of the chip.
- The rest are either not-used in this project or NC.



*Image 4. 1 LCD pinout*

The pin connections between the LCD board and the processor can be found in Table 4.1.

LCD	Processor Board
VCC	5V
GND	GND
DB0...15	Port C pin0...15
RS	Port A pin0
WR	Port A pin1
CS	Port A pin2
RST	Port A pin3

*Table 4. 1 LCD to Processor pin connection*

All these pins are configured as output push-pull with a frequency of 50MHz

The HX8357 accepts 69 different commands, but only a few of them will be useful in this project (found in annexes 4.1 to 4.4). To send commands the microcontroller, the right sequence of actions must be completed:

- 1) Enable the controller (LCD\_CS=0).
- 2) Tell the controller it's about to receive a command code. (LCD\_RS=0).
- 3) Make LCD\_WR=0 to prepare the rising edge.
- 4) Put the command code in parallel on the DB0—DB7 pins (command codes are only 8bit long).
- 5) Apply the rising edge on LCD\_WR (set it from 0 to 1) so the microcontroller knows that there is useful data in DB pins.
- 6) Disable chip communication and command transmit. (LCD\_CS = 1 and LCD\_RS =1).

Some commands need to receive one or more arguments through the DB pins after the command. For example, after the RAMWR command (a command that writes data into the microcontroller's RAM) the microcontroller needs to receive the data that is going to write on its RAM. That is made with a consecutive data transmit with a similar structure:

- 1) Enable the controller (LCD\_CS=0).
- 2) Tell the controller it's about to receive data regarding the previous command. (LCD\_RS=1).
- 3) Make LCD\_WR=0 to prepare the rising edge.
- 4) Transmit the configuration data in parallel on the DB0—DB15 pins (some arguments are 16 bit long).
- 5) Apply the rising edge on LCD\_WR (set it from 0 to 1) so the microcontroller knows there is useful data in DB pins.
- 6) Disable chip communication. (LCD\_CS = 1)

Note that in the next section the following agreement will be used:

- LCD in horizontal position with the pins on the right. Like image 4.1
- Origin of coordinates is top left corner
- There are 320 rows and 480 columns

## Initializing the LCD

Before utilizing the LCD it must be configured in the right way. There are multiple ways one can configure this LCD depending on the requirements of the application. Which configuration is chosen for this project and why is detailed in the following.

One of the first things one has to choose is how the LCD is going to update its content. There are two interfaces to choose from:

- DPI (Data Pixel Interface): in this interface the user has to stream continuously the pixels to the data bus. To synchronize the pixel position on LCD with the data being received two sync signals are used, Vsync and Hsync. This interface offers faster



change rate of big chunks of pixels but has the disadvantage of needing a dedicated graphic processor to handle the continuous data transfer.

- DBI (Data Bus Interface): The values of the pixels are stored in the LCD microcontroller memory. Each memory position contains data of one pixel and with the refresh frequency of the LCD (this value can be configured) the LCD microcontroller takes these values and print them on the screen. To modify the contents of the screen data must be written on the microcontroller memory and in the next refresh the LCD will show the updated image.

This interface is perfect for images with few moving parts or sudden changes; because writing in the memory the combination of pixels is enough, the LCD microcontroller will take care of the rest.

On the other hand if it's needed to change a lot of pixels at once (sudden image change or big area movement) this interface can lead to slow refresh rate or even visual glitches. That is because the writing of big areas of memory can take longer than the refresh period; this causes a full image update to take more than one refresh to complete.

In fact DBI is actually a DPI that gets its continuous data stream from internal memory. In this interface the LCD microcontroller let us choose between giving an external Vsync and Hsync signals and using its internal clock for that matter.

Note that there are two types of DBI interface; type B, which uses parallel transmissions, and type C, that uses serial transmission.

The chosen interface was the DBI interface (we are forced by board manufacturer to use type B). Having the processor occupied sending a continuous stream of bits to the LCD was not an option, the processor needs to still have time to evaluate what is going on in the game. For this same reason the internal clock of the microcontroller was chosen for the sync.

The necessary command to configure this is SETDISPLAY. First, it's necessary to send to the processor the command code (0xB4) and then the configuration data. As it can be seen in image 4.2, to chose the DBI interface and the internal oscillation clock RM bit must be 0 and DM bits bi 00, so after the command transfer we need to do a data transfer with the chosen configuration (0x00);

B4 H		SETDISPLAY											
	DCX	RDX	WRX	D17-D8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
Command	0	1	↑	-	1	0	1	1	0	1	0	0	B4
1 <sup>st</sup> parameter	1	1	↑	-	0	0	0	RM	0	0	DM[1:0]		XX
Description	<b>RM:</b> The bit is used to select an interface for the Frame Memory access operation. The Frame Memory is accessed only via the interface defined by RM bit. Because the interface can be selected separately from display operation mode, writing data to the Frame Memory is possible via system interface when RM = 0, even in the DPI display operation. RM setting is enabled from the next frame. Wait 1 frame to transfer data after setting												
	RM		Interface for RAM Access										
	0		DBI Interface (CPU)										
	1		DPI Interface (RGB)										
	<b>DM[1:0]:</b> The bit is used to select display operation mode. The setting allows switching between display operation in synchronization with internal oscillation clock, VSYNC, HSYNC or DPI signal.												
	DM 1		DM 0		Display Mode								
	0		0		Internal oscillation clock								
	0		1		External VSYNC + HSYNC ( Display data from GRAM)								
	1		0		External VSYNC ( Display data from GRAM)								
	1		1		External DPI ( RGB Through mode)								
	Note: Switching between VSYNC, HSYNC and DPI operation is prohibited.												

Image 4. 2 Set display configuration

The next thing to define is how the pixel information will be passed to the LCD controller.

Note that in the microcontroller memory each pixel is represented by 18 bits (each colour's intensity is defined with 6 bits). So for each pixel the microcontroller has to receive 18 bits.

But the board that includes the microcontroller and LCD is designed with a 16 bit data bus (the controller is prepared for different data bus sizes). The problem comes here; it's needed to transmit 18 bit information through a 16 bit bus. This would force the processor to make more than one transmission per pixel slowing the data transfer. There is an alternative though; the LCD controller can use other pixel formats, configurable with the command 0x3A, SET\_PIXEL\_FORMAT.

In image 4.3 can be found the structure of this command.

- D4,D5,D6 define the pixel format for the DPI interface
- D0,D1,D2 define the pixel format for the DBI interface

3A H	COLMOD (Interface Pixel Format)												
	DCX	RDX	WRX	D17~D8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
Command	0	1	↑	-	0	0	1	1	1	0	1	0	3A
1 <sup>st</sup> parameter	1	1	↑	-	X	D6	D5	D4	X	D2	D1	D0	XX

Image 4. 3 SET\_PIXE\_FORMAT command structure

This microcontroller has 3 pixel formats that can be selected:

- 3 bit pixel (value = 001) 8 colors. Only for DBI type C interface
- 16 bit pixel (value = 101) 64k colors
- 18 bit pixel (value = 110) 256k colors

For the 16bit pixel format, the data bus is used as can be seen in image 4.4

Register	DB17	DB16	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Command
Set_pixel_format	DFM	x	x	x	x	x	x	x	x	x	0	0	1	0	1	1	1	0	2EH
3'h5	X	x	x	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0
65K-Color (1-pixels/ 1-transfer)																			

Image 4. 4 16bit pixel format transfered in 16bit data bus

In this format, instead of having 6 bits for each colour, there are 5 bits for red, 6 bits for green and 5 bits for blue. Now the LCD microcontroller needs to transform this 16 bit information into 18 bit information so it can save it on its memory. The green information is already complete so it is only necessary to extend the red and blue data. This can be accomplished in 3 different ways, detailed in image 4.5 :

- Make 1 shift left, padding with 0
- Make 1 shift left, padding with 1
- Make 1 shift left, padding with the MSB

Register		GRAM Data																	
Set_pixel_format	EPF[1:0]	DB17	DB16	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
3'h5	2'h0	R4	R3	R2	R1	R0	0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0	0
	2'h1	R4	R3	R2	R1	R0	1	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0	1
	2'h2	R4	R3	R2	R1	R0	R4	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0	B4

Image 4. 5

The method used for extending the 16 bit info depends on the value of EPF, configured using the 0xB3 command (check annex 4.2 for more information).

In this project the default value of EPF was used because it was considered that the effect of the LSB in a 6 bit colour resolution was irrelevant.

Note that other alternative would have been to chose 18 bit pixel format and make 2 data transmissions per pixel as seen in image 4.6, but in behalf of speed and not overloading the processor the above method was chosen.

Register	DB17	DB16	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Command
Set_pixel_format	DFM																		2CH
3'h6	0	x	x	R05	R04	R03	R02	R01	R00	x	x	G05	G04	G03	G02	G01	G00	x	262K-Color (2-pixels/ 3-transfer)
		x	x	B05	B04	B03	B02	B01	B00	x	x	R15	R14	R13	R12	R11	R10	x	
		x	x	G15	G14	G13	G12	G11	G10	x	x	B15	B14	B13	B12	B11	B10	x	
	1	x	x	x	x	x	x	x	x	x	x	R05	R04	R03	R02	R01	R00	x	262K-Color (1-pixels/ 2-transfer)
		x	x	G05	G04	G03	G02	G01	G00	x	x	B05	B04	B03	B02	B01	B00	x	
		x	x	x	x	x	x	x	x	x	x	R15	R14	R13	R12	R11	R10	x	

Image 4. 6 two different ways of transferring 18bit pixel frame through a 16 bit bus.

The next thing necessary to choose is how the microcontroller's memory will be used by the display; we will do that with the SETPANEL command. In image 4.7 the command structure can be found.

C0 H	SETPANEL												
	DCX	RDX	WRX	D17-D8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
Command	0	1	↑	-	1	1	0	0	0	0	0	0	C0
1 <sup>st</sup> parameter	1	1	↑	-	0	0	0	REV	SM	GS	0	0	XX
2 <sup>nd</sup> parameter	1	1	↑	-	0	0	NL[5:0]						XX
3 <sup>rd</sup> parameter	1	1	↑	-	0	SCN[6:0]							XX
4 <sup>th</sup> parameter	1	1	↑	-	0	0	0	NDL	0	PTS[2:0]			XX
5 <sup>th</sup> parameter	1	1	↑	-	0	0	0	PTG	ISC[3:0]				XX

Image 4. 7

In the first parameter:

- REV: refers to the colour polarity. If its value is 1 normal colour is selected, otherwise it will print the image's negative on the screen. We will set it to 1

- SM: scan mode. This will make the microcontroller to read its internal memory in ascending or descending order. The LCD output will be the same, this is an option for code optimization in some special cases, and in this case it's irrelevant.
- GS: gate select. This will make the microcontroller to scan all memory values in order or to read first the odd memory positions and then the even ones. Like SM this is for code optimization, irrelevant in our case.

Second parameter:

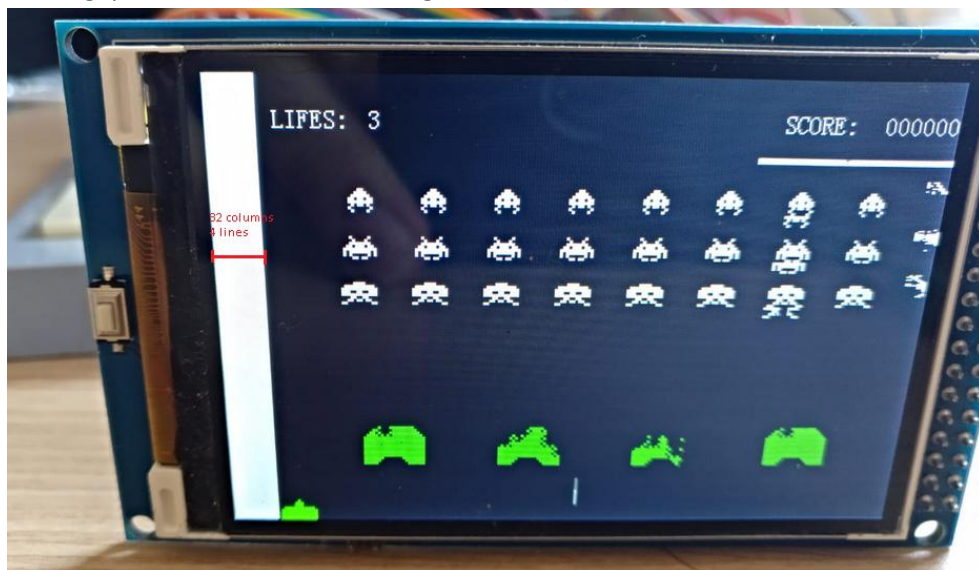
- NL: Sets the number lines that will be displayed on the LCD. Each line equals 8 columns. The columns displayed are calculated like this:

$$\text{Columns displayed} = 8 * (\text{NL}[5:0] + 1)$$

Our LCD has 480 columns so NL can't be greater than 0x3B (59 in decimal, this value makes columns displayed equal 480).

If set anything under 0x3B the image shown will have a band of columns in white (colour by default, can be changed). After that band the LCD will start displaying the contents of its memory as normal.

In image 4.8 the effect of setting the NL to 0x37 (55 in decimal) can be seen. This causes a gap of 4 lines (each line being 8 columns).



*Image 4.8 Gap of 32 columns after setting NL to 0x37*

In this project the chosen NL was 0x3B to use all LCD pixels.

Third parameter:

- SCN: this sets the starting scan position. The starting memory position is calculated like this:

$$\text{Starting position} = 1 + \text{SCN}[6:0] * 4]$$

This causes that the image shift left, for example, for SCN=0x04, starting position will be 17, so in column 1 we will have what otherwise would be column 17.

This is useless in this project so we will SCN value will be 0x00.

Fourth parameter and fifth parameter:

This values configure aspects of the band that appears when NL has a value<0x3B in NL, irrelevant in this project.

Now the only remaining thing is to exit sleep mode (the display enters it after a reset) and set the display on (so the data from the memory is actually displayed on the LCD).

For that the commands are:

- Exit\_Sleep: 0x11, no parameter needed.
- Wait for about 100ms to exit sleep.
- Display\_On: 0x29, no parameter needed.

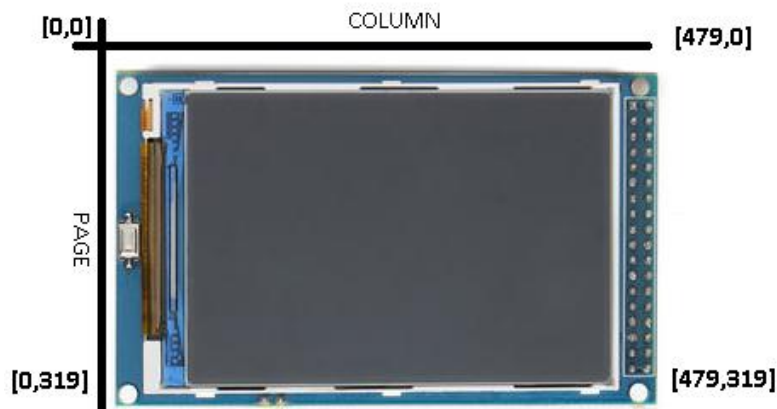
## Displaying images in the LCD

After the display is initialized with the correct configuration it is ready to start displaying images.

To do so, the microcontroller has a set of instructions that helps us to change only a small part of the display instead of having to update all the pixels. With the following commands it is possible to define an “update window”. This makes that only a small region of the memory will be written instead of all the LCD:

- CASET (0x2A Column Address SET) needs the starting and ending column address that will be edited as attributes. Column address is 16 bits long but this command only takes 8 bit data so, in total, this will need 5 data transfers (1 command code, 2 data transfers for starting address, 2 data transfers for ending address) the address transfers follows big endian (Most significative byte first)
- PASET (0x2B Page Address SET) needs the starting and ending page (or Row) address that will be edited as attributes. The transfer structure is the equal to CASET.

Following the reference system in image 4.9, and this commands it is possible to define a small windows to edit only that region, see the following example.



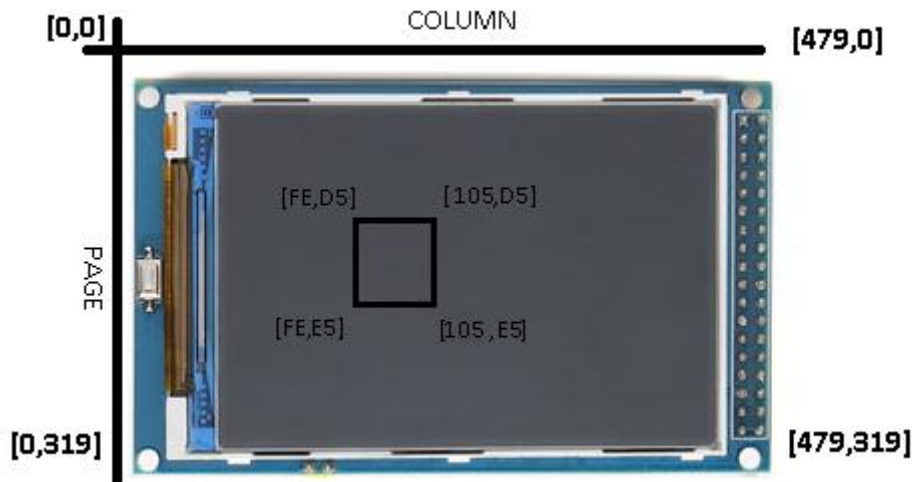
*Image 4. 9 reference system*

Example 1:

To update a small part of the LCD, drawing a small image of 16x16 starting at [254, 222], the window will extend columns 0xFE (254 in hex) to 0x10E (270 in hex) and pages 0xD5 to 0xE5. So the data transfers must be the following:

- Command1: CASET
  - o Transfer 1, command code: 0x2A
  - o Transfer 2, high byte of start address: 0x00
  - o Transfer 3, low bit of start address: 0xFE
  - o Transfer 4, high byte of end address: 0x01
  - o Transfer 5, low bit of end address: 0x0E
- Command1: PASET

- Transfer 1, command code: 0x2B
- Transfer 2, high byte of start address: 0x00
- Transfer 3, low bit of start address: 0xD5
- Transfer 4, high byte of end address: 0x00
- Transfer 5, low bit of end address: 0xE5



*Image 4. 10 windows creation example*

The next step is to send the actual pixel data. This is done using the command RAMWR (0x2C RAM WRite). After the command code is send, each following data transfer will write the pixel data on its corresponding address of microcontroller memory.

Following the position and reference system in Image 4.9, the first data transfer will write the pixel on the top left corner of the last defined “update windows”, the next transfer will write the pixel on the right of the last one, once the entire first row has been written the processor focus on the first pixel of the next row automatically. This way, consecutive transfers write data on the right positions in the window without having to implement more code, just sending all pixel data one after another in the right order is enough.

Example 2:

To draw a small area (4x4) with a chess like pattern the sequence of commands will be the following:

- PASET to work on the intended area, in this example from (0x0000 to 0x0003)
- CASET to work on the intended area, in this example from (0x0000 to 0x0003)

This configures the screen as seen in image 4.11.

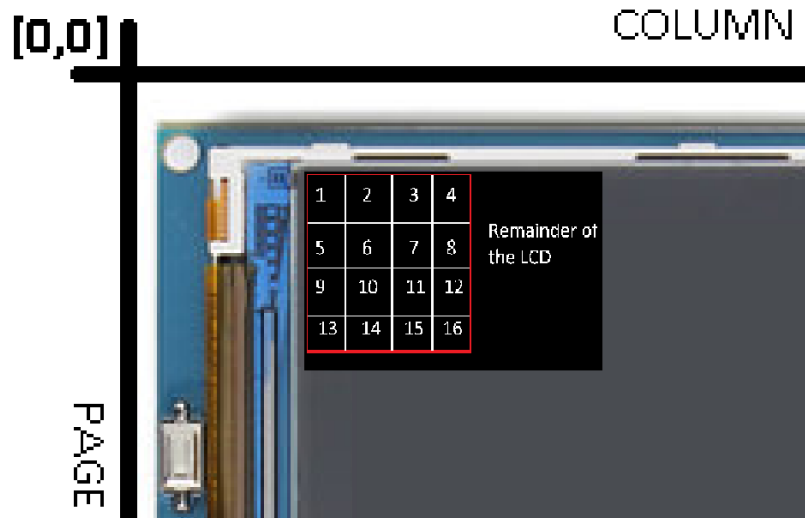


Image 4. 11 update window of example 2

- RAMWR:
  - o Send command code 0x2C
  - o Send color for pixel 1: black (0x00)
  - o Send color for pixel 2: white (0xff)
  - o Send color for pixel 3: black(0x00)
  - o Send color for pixel 4:white (0xff)
  - o Send color for pixel 5: white (0xff)
  - o ...
  - o Send color for pixel 16:black (0x00)

The final result will be:

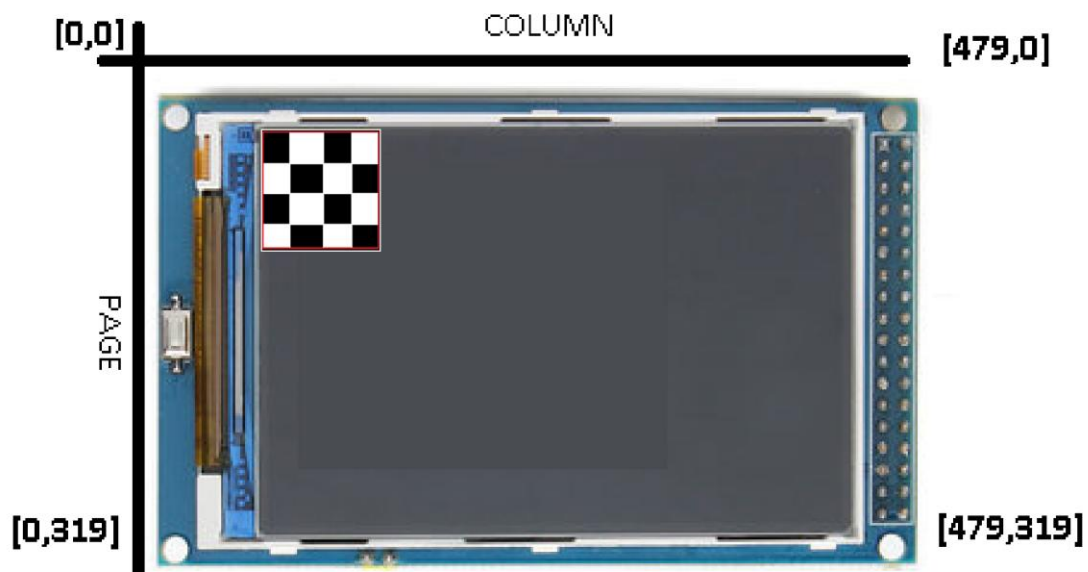


Image 4. 12 display of example 2

The rest of the screen will be unaltered.



Because of this behaviour (pixel data being transmitted as a single stream regardless of being in different column/rows) the easiest way to manage image/pixel data is through 1 dimensional arrays. This will be detailed in the game code section.

## 5. Loading games

As a gaming platform, G-Engine needs to be able to load different games. It would not be wise to invest in a platform that can only play a game that is loaded on manufacturing.

In order to accomplish that goal there are different approaches but the ones that fit more with this project are the following:

- Connect the platform to a PC via USB and load the game into its internal memory.
- Load the game to an SD card and then make the processor load it to its internal memory from the card.

Though at first it may seem that the second option is harder and involves more steps after some analysis it was the chosen solution. Here is why:

In order to connect a device via USB to a computer, it must implement flawlessly the USB protocol, and it's not an easy protocol to implement. Another problem will be the internal storage, although this board has a lot of internal memory it will hardly be enough to store more than one complex game with average graphics. And even if tried to load several lower spec games, there's the problem that not all games occupy the same quantity of memory making it impossible to work with predefined locations for code execution and forcing the implementation of a variable location code.

On the other hand a system running with an SD card reader has several advantages.

- The user can carry several SD cards with different games, and changing from one game to another would be as easy as changing the SD card. This is great for a portable device because you can change the game without needing a computer
- The capacity of the SD cards is far superior to the internal memory of our processor, making it possible to have more than one game on one SD card.
- The USB protocol and the file system are already implemented, simplifying a lot the project.

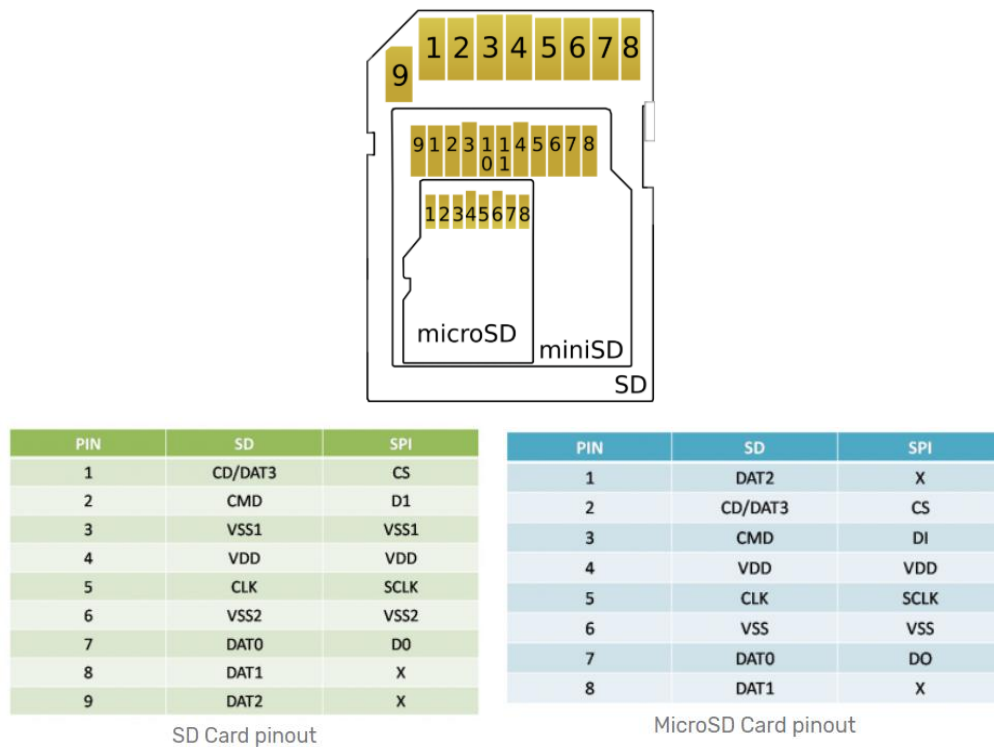
So the loading method chosen is the SD card. Let's see how that works.

## Communicating with the SD card

Depending on the format of the SD card the pin out is different (see image 5.1). This document will focus on the micro SD card.

The micro SD card has 8 connection pins and their purpose depends on the interface used to communicate with the card (Image 5.2 and 5.3). There are two possible interfaces: the SD interface and the SPI interface. For simplicity, the interface chosen in this project was the SPI. So the pin out is the following:

- CS chip select. Part of the SPI interface
- MOSI. Master out slave in. The Data in for the SD card.
- SCLK. Synchronization clock. Necessary for the SPI interface.
- MISO. Master in Slave out. The Data out for the SD card.
- VDD. Power input, 3,3V
- VSS. Power input, GND
- 2 pins are left unconnected since they are only for the SD interface

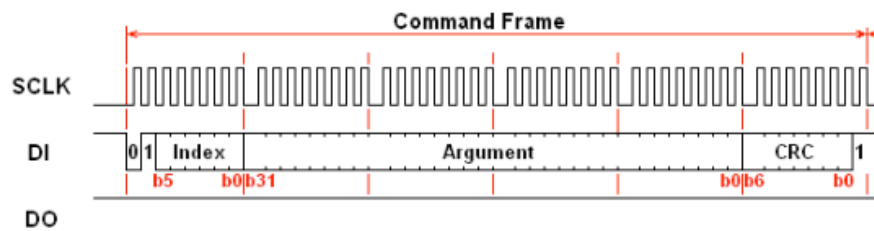


*Image 5. 1 pin out of different SD cards formats*

The slave (the SD card) and the master (the processor) will communicate through SPI using frames. There are three types of frames and they have different structure.

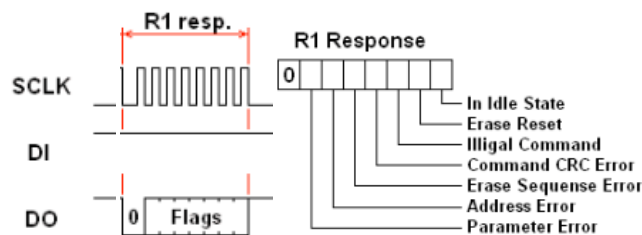
- Command frame: Send by the master. Consist of 6 bytes (Image 5.2):
  - o Byte 1 Command code. It will always start with 01 followed by the command number. All commands can be found in Annex 5.1 to 5.3.

- Byte 2-5 arguments for the command. These bytes must be sent even if the command doesn't need arguments. In that case just send 0xFF
- Byte 6 CRC. Security check to ensure that the transmission has carried out without errors



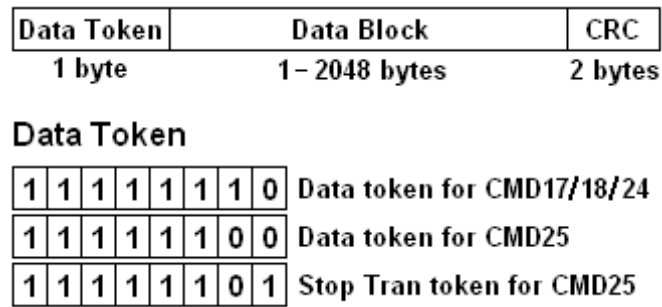
*Image 5. 2 Command frame structure*

- Response frame: sent by the slave (Image 5.3). Generally 1 byte long (except for some commands that send information back to the master) consists in a series of flags that indicate the internal status of the SD card.
  - Bit 7. This is always 0
  - Bit 6. Parameter error flag. Parameter of the last command is not valid.
  - Bit 5. Address error flag. Address of the last command is not valid.
  - Bit 4. Erase sequence error flag.
  - Bit 3. Command CRC error flag. The CRC check indicates an error during transmission.
  - Bit 2. Illegal command flag. Command not valid
  - Bit 1. Erase reset.
  - Bit 0. In idle state. In this state the SD card is waiting for initialization, only a few commands are valid.
  - (Optional) 4 bytes of data. Some commands require the SD card to send some SD internal registers. They come immediately after the response byte



*Image 5. 3 Response frame*

- Data frame (image 5.4): Send by the slave after a read memory command. The content of the specified address of memory is sent to the master. Consists of a maximum of 2051 bytes:
  - Byte 1: Data token. Varies depending on the command that caused the transmission (single block read, multi-block read or stop read)
  - Byte 2-2049: Data block. Content of the memory. Its length depends on the sector size. This can be changed in the initialization process.
  - Byte 2050-2051 CRC



*Image 5. 4 Data frame*

## Initialize the SD card

The SD card needs about 1ms after power up before it is ready to receive commands. After that, to select the SPI interface, the SD card needs to receive at least 74 clock periods with the CS and MOSI pins HIGH. This will be achieved by sending 10 bytes containing 0xFF. Note that the clock frequency in this stage must be between 100 KHz and 400 KHz.

Now the SD card is ready to receive commands through SPI, but after reset the card begins in idle state, this means that only a few commands are accepted and the ones that read data from internal memory are not part of them. To exit idle state the following sequence of commands must be sent.

Send Cmd0 with CS low, no arguments and the correct CRC (0x95). This resets the internal configuration of the SD and sends a response frame that needs to be 0x01 (meaning it's on idle state and no errors have been found). If it returns error or no response we can't initialize the card, its type is unknown. As CRC default value is false, from this point on the CRC byte of the command frame can be any value since it will not be checked. It can be activated using command 59. CRC is a number calculated from some bits of the frame and helps to know if there has been any defections in the transfer.

After that we have several initialization paths depending on the SD card version. For the purpose of this project, versions are only relevant as to the initialization process. The main differences between them are transfer speed and maximum capacity.

Version 1 has capacities up to 2GB, version 2 up to 32GB and version 3 up to 2TB.

To know the version of card CMD8 with argument 0x1AA is send. This command is illegal in version 1 SD cards. This way, if in the response the illegal command flag is set, the card connected is a version 1 card. Note that CMD8 is one of the commands with extended response frame; this means that before sending the next command the processor needs to receive the four optional bytes in response frame. Note also that this response's last 12 bits must be equal to the argument sent, in this case 0x1AA. If the response does not end with that value an error happened or the card is not an SD card.

CMD8 original purpose is to check the voltage range admitted by the card but in this project will be used only to know the card version since the voltage regulation is handled by the SD card to SPI adapter. Next steps are different depending on the version of the card.

**For version 2 or higher cards**

Send CMD55 to access extended commands. Now we send CMD41, this is going to start the initialization of the card, this doesn't need arguments either, this process may take as long as 1 second. To know when the process is over, CMD41 must be continuously re-send until the respond frame show that the card is no longer in idle state (response frame byte must equal 0x00).

**For version 1 cards**

The command to initialize the card in this version is CMD1 instead of advanced command CMD41. The following steps are the same as seen in V2, resend CMD1 until response is 0x00 or the device takes longer than 1 second, which will indicate that something is wrong with the card.

Now that the card is initialized, in order to assure compatibility with FAT32 an extra step is necessary. The block size for read and write operations must be 512 bytes (this will affect the data frame length). In case the default value for block size is not 512, CMD16 must be used to change it. It's as simple as sending CMD16 with its argument being 0x200 (512 in decimal).

After the initialization is completed is recommended to change the SPI clock frequency, the previous limit was only to initialize the card, now it can handle faster baud rates. To know the maximum frequency of transmission the CMD9 is used (with no argument). This will trigger a data transmission, after the habitual response frame, of an internal register named CSD.

The CSD register holds a lot of useful information about the SD card; CDS has 16 bytes and its structure can be found in Annex 5.4. The TRAN\_SPEED field holds the maximum clock frequency supported by the car. It's located on byte 12 (bits 96 – 103) of the send data packet, and follows the structure found in image 5.5:

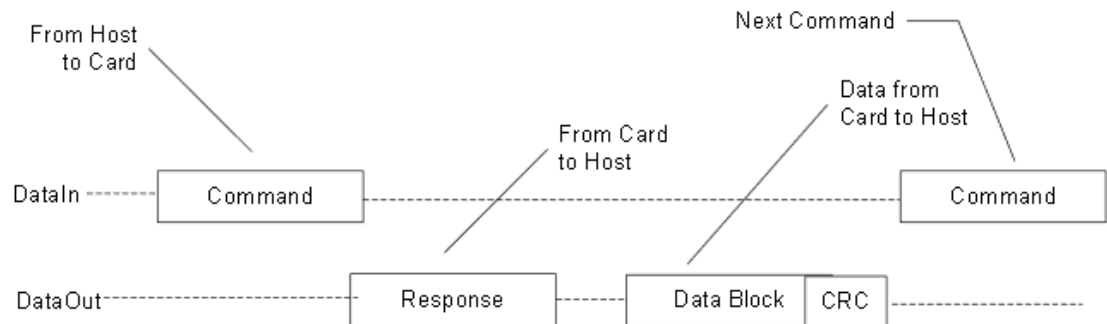
TRAN_SPEED Bit	Code
2:0	transfer rate unit 0=100kbit/s, 1=1Mbit/s, 2=10Mbit/s, 3=100Mbit/s, 4... 7=reserved
6:3	time value 0=reserved, 1=1.0, 2=1.2, 3=1.3, 4=1.5, 5=2.0, 6=2.5, 7=3.0, 8=3.5, 9=4.0, A=4.5, B=5.0, C=5.5, D=6.0, E=7.0, F=8.0
7	Reserved

*Image 5. 5 TRAN\_SPEED byte structure*

After adjusting the speed of the SPI interface for our processor to match the maximum allowed by the card, the last step remaining is how to read the SD internal memory.

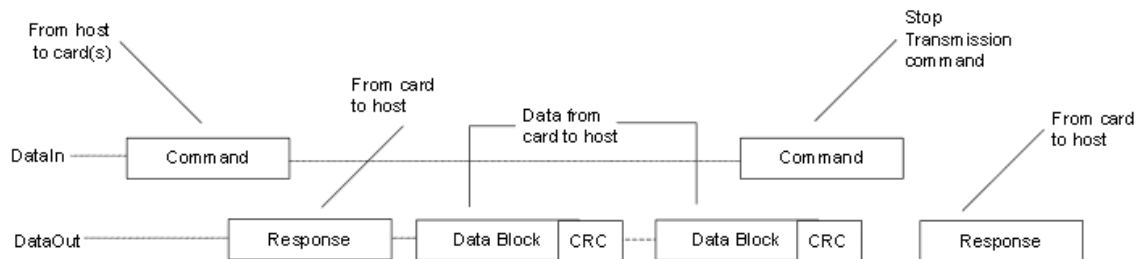
There are two commands involved in data transmission that will be relevant for this project.

- CMD17: Read single block. Reads the data block in the address specified in the argument. Image 5.6



*Image 5. 6 CMD17 command response*

- CMD18: Read multiple blocks. Starts reading data blocks at address specified in the argument until end of sector or receiving CMD12 stop transmission (no argument)



*Image 5. 7 CMD18 command response*

## FAT32 file system

In order to ease the transfer of the games to SD card, the best option is to maintain the file system used on PCs and teach our processor to gather data from a volume that implements that file system.

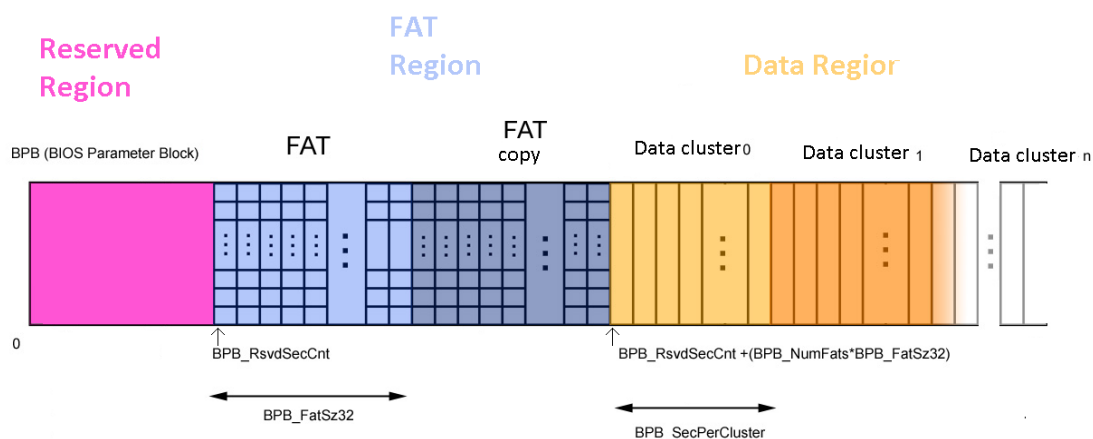
A file system is an interface between the Operating system and the volume, in this case the SD card. A file system organizes the information in the volume and optimizes the space usage and reading speed.

The file system present in the vast majority of volumes is FAT32, developed by Microsoft. FAT stands for File Allocation Table, because the memory allocations of all the files and directories of the volume are stored as a table in a region of its memory.

In the FAT32 file system there are 3 regions. A simplified memory mapping can be seen in image 5.8:

- BPB (BIOS Parameter Block).
- FAT region
- Data region

Each region is composed by clusters, and clusters are formed by sectors; sectors are groups of 512 bytes and are the reason why the SD card needs to be configured to read blocks of that size. Clusters, on their side, are composed by a number of sectors that depend on the volume. The actual number of sectors in a cluster can be found in one of the fields in the BPB region. Since this file system is oriented to high capacity volumes the allocation is made within clusters. That means that in a volume with 4 sectors per cluster there are  $4 \times 512 = 2048$  bytes between two consecutive addresses.



*Image 5. 8 Fat data structure. Image extracted from [www.tolaemon.com](http://www.tolaemon.com) and modified by myself*

### **BPB**

It has all the parameters of the FAT volume. Volume size, number of sectors, FAT size, FAT starting sector... all can be found in the annex 5.5. It starts at address 0.



The more important fields in this region are:

- BPB\_RsvdSecCnt: indicates the starting sector of the FAT table
- BPB\_FatSz32: indicates the size of a single FAT table, commonly there is more than one copy
- BPB\_NumFats: number of copies of the FAT. In conjunction with Fat size and Fat starting sector helps to locate the start of Data region
- BPB\_SecPerCluster: indicates how many sectors of 512 bytes are in a cluster.
- BPB\_RootClus: indicates the starting cluster of the Root directory. The parent directory of all other directories

## **FAT**

FAT stands for File Allocation Table.

The FAT is just a series of 32 bit entries that contains cluster addresses. The number of entries in the table will depend on the number of cluster the data region has since the FAT has one entry per cluster.

This tables map how a file is distributed in the several clusters. When a file is saved in a FAT volume its contents are chopped in pieces and are, then, saved into available clusters in the volume. But these clusters are not necessarily consecutive and because of this the FAT table exists. In the FAT table there is an entry for every cluster in the volume, and that entry contains the cluster number of the next file piece.

Imagine a file saved in a FAT volume. In the data region the only information easily accessible is the cluster where the file's first piece is stored.

For example: In a FAT32 volume there is a file which first cluster is cluster 13. In the corresponding FAT entry (entry 13) there is stored the value 15. The next step is to look at position 15 of the FAT table, containing number 20. Finally in position 20 of the FAT table there is the value 0xFFFFFFFF, meaning that is the last sector containing data for that file (any number greater than 0xFFFFFFFF7 means that it's there are no more clusters containing information of that file).

So, to reconstruct the file is necessary to concatenate the data found in cluster 13, 15 and 20, in that order.

## **DATA**

This region is divided in clusters. There are two types of clusters, those containing file data and those containing directory data.

In the FAT system, directories' data structure consists in a series of 32 byte entries. Each one of these entries can belong to either another directory or a file. Note that every directory will have as first entry its parent folder (this doesn't apply in the root directory).

Of all the directory entry fields, this project will only use: DIR\_Name, DIR\_Attr and DIR\_FstCLus. The rest can be found in the Annex 5.6:

The DIR\_Name field consist in 11 bytes and contains the short name of the file or directory, each byte correspond to an ascii character.

The DIR\_Attr byte indicates, among other things, whether the entry corresponds to a directory, a file or a special entry, the long name. If it is a file this byte will start with 0x2, if it is a directory it will star with 0x1 and if it's a long name this byte's value will be 0x0F. The other parameters are not necessary for this project but they are self-explanatory, Read-only, Hidden...

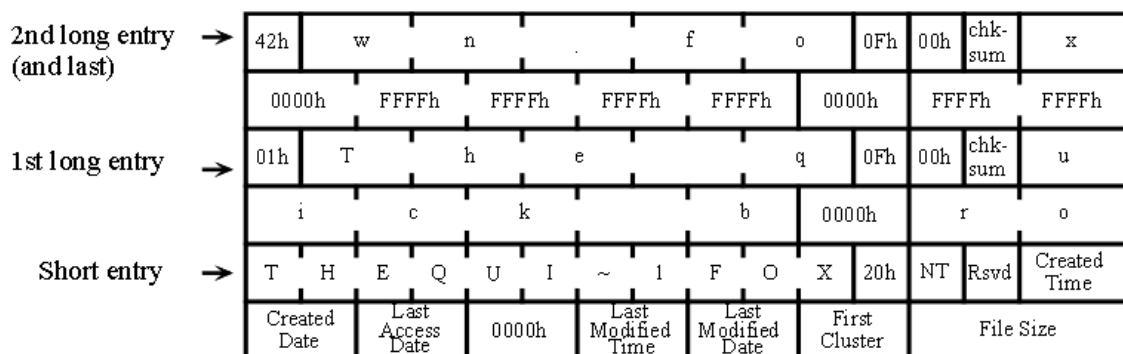
Apart from the attribute byte, in this project the most important fields are DIR\_fstCluHI and DIR\_fstCluLO, which says the cluster number where the file or directory starts.

Note that in the directory entry the name can only be 11 characters long; the name in this field is, in most cases, a reduced version of the original name. If a directory entry have a name longer than 11 characters (including the file extension) a special entry is used. This entry is neither a file nor a directory; it's only a part of a long name. This entry, or entries, goes before the directory entry to which it belongs. Each long name entry contributes with 13 extra characters, and apart from the characters themselves the only important fields are:

- LDIR\_Ord: indicates the order of this entry in the sequence of long name entries. If masked with 0x40, this indicates the entry is the last of the long name entry set.
- LDIR\_Attr: as seen before this identifies the entry as a long name if this field value is 0x0F

Detailed structure of this entry can be found in Annex 5.7

Suppose a file is created with the name: "The quick brown.fox". Image 5.9 illustrates how the name is packed into long and short directory entries.



*Image 5. 9 Example of a long name*

Note how the first entry found in the memory corresponds to the last entry of the directory entry, FAT32 follows big endian structure. So the first entry found identifies itself as a long dir (ATTR =0x0F), it also says that is the last long dir of a set of 2 (has the LAST\_DIR mask and the number 2 for the order). Next to it is found the 1<sup>st</sup> long dir entry (LDIR\_ORD=01), and finally the short dir entry. The short dir corresponds to a file (Attr starts with 0x2).

Note that every folder (except for the root directory) has a directory entry that point to the parent folder.

### **Retrieving data from a FAT32 volume**

Using the read command of the SD card we retrieve the first sector of data (SD read sector 0), in the FAT32 file system this sector contains the volume information and characteristics. The important fields are:

- BPB\_RootClus: starting cluster of the root directory.
- BPB\_SecPerClus: number of sectors in a cluster.
- BPB\_reservedSecCnt: this specifies the sector number where the FAT table starts

Next step will be going to the root directory starting sector and list all its entries. This is accomplished using the BPB\_rootClus from the BPB region, this field indicates the cluster number where the root directory starts. Then using the FAT the processor can know if there are more clusters and where are they.

To know in what sector a specific entry of the FAT is located just apply the next formula.

$$\text{Sector number} = BPB_{reservedSecCnt} + N * secPerClus * entry\_size / sector\_size$$

N is the cluster number whose entry we want to know (RootClus in this example), sePerClus is the number of sector in each cluster, entry\_size is the number of bytes a FAT entry has (in FAT32 always 4) and sector size which also is always 512.

This is necessary when reconstructing a file or directory: usually only the first cluster of a file can be found, but to find the next cluster it's necessary to go to the FAT. The problem is that the processor can't access all FAT at once, it can only ask for sectors of 512bytes to the SD card. And to know which sector to ask in order to get to the next cluster the processor uses the above formula.

Now, to know specifically where in that sector will be the entry, the following formula needs to be used.

$$byte\_offset = (N * 4) \% 512$$

Where  $a \% b$  is the remainder of the division  $a/b$ .

Note that the SD addresses sectors while the FAT addresses clusters. Once the cluster number that needs to be accessed is known, is important to translate that cluster number into sector number in order to retrieve the correct information from the SD card. This is as simple as:

$$sectorNumber = clusterNumber * BPB\_reservedSecCnt$$

With the directory reconstructed, all the files and directories inside of it are accessible and can be listed using its name and used using the fields DIR\_fstCluHI and DIR\_fstCluLO which point to the element's first cluster.

## 6. About the code

Once the console is turned on, and after the few milliseconds that the processor needs to initialize, the code loaded into the microcontroller's flash memory starts. Later, when a valid SD card is connected and a game in there is selected, program execution will jump to the game's code.

In this section will be explained what that code does and how. All the code can be found in the annexes 6.1 to 6.14, in this section all will be explained using simple flowcharts and comments whenever is necessary.

Keil has the ability to export a file containing all the code in machine language. The game file the processor needs to retrieve from SD card and copy into its memory is precisely this file.

### 6.1 Main program

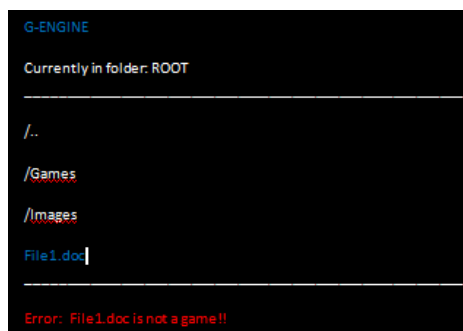
This code is the one loaded in the microcontroller from the beginning. This needs to enable and prepare all the peripherals that will be used in detecting and reading the SD card, show images on the LCD and reading the buttons pressed in the controller module.

It also needs to understand the contents of the SD card, tell if a file is a valid game or not and, for valid games, load its code into memory and execute it.

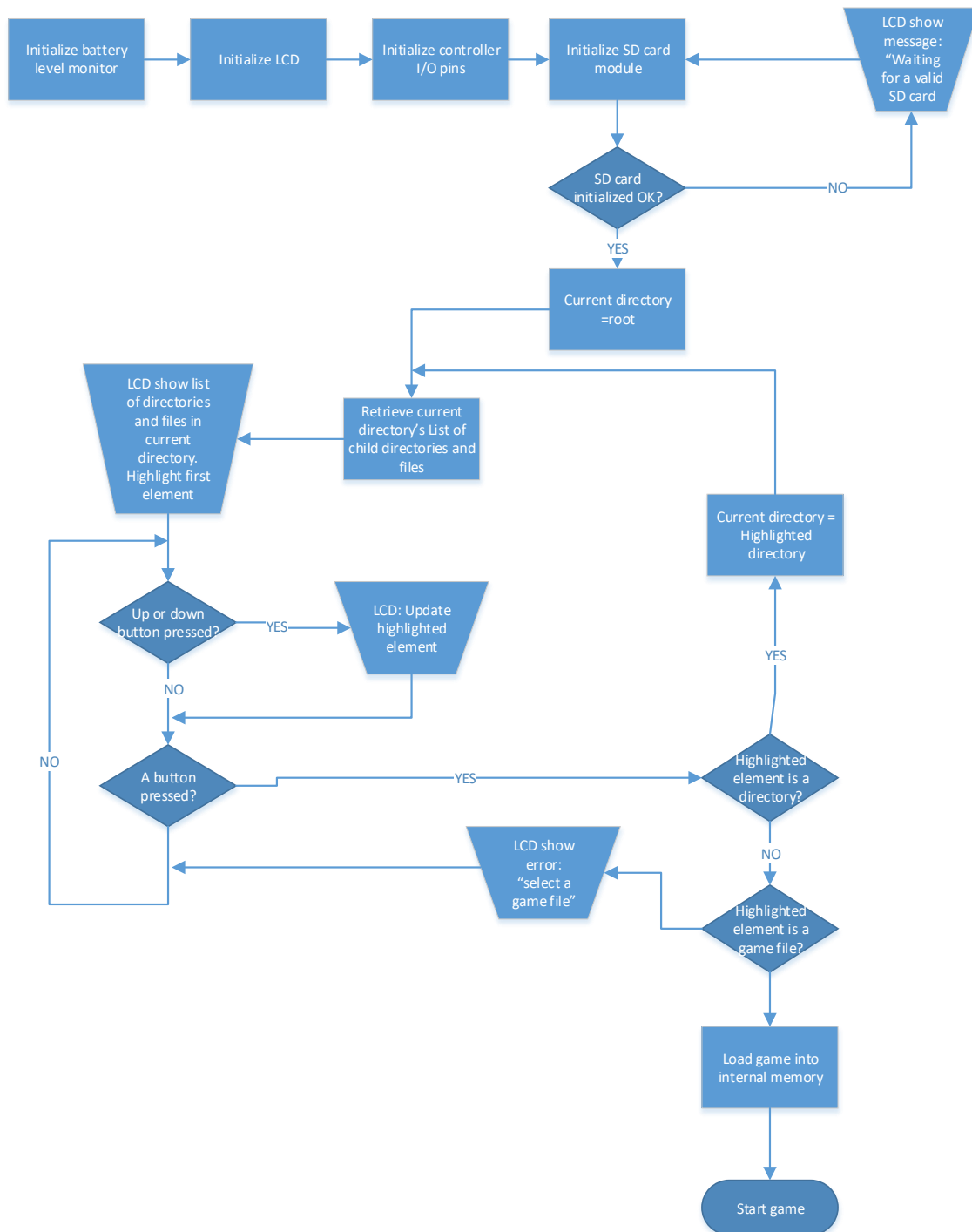
After initializing all the modules (SD card, LCD, controller and battery monitor), the program accesses the SD card and shows a list of all the files and directories contained in the root directory on the LCD. If the SD card module could not initiate (either no SD card or an invalid one is connected) a message appears on the LCD indicating to connect a valid SD card and retry the initialization. With all the directories and files listed on the LCD the first element will be highlighted, shown in another colour. In Image 6.1 can be seen how the LCD shows the list of directories.

To navigate through the directories the controller is used, up or down to highlight a different elements and "A" button to confirm selection. If the element selected is a directory, the process repeats. If it's a game file, the code is loaded and executed. If it's other kind of file an error message is shown.

This code also checks periodically the state of the battery to update the LED in case of low battery. The flow chart of this code can be found in image 6.2



*Image 6. 1 Directories list format. File 1 is highlighted*



*Image 6. 2 Main program*

## 6.2 Retrieve SD card directories.

The details of some of these operations are explained in section 5. Here, this document focuses on explaining the flowchart in general that can be found in image 6.3.

The program starts with the root directory first cluster number. It converts this cluster number into sector number and asks for that sector to be transferred by the SD.

After that, the code checks all that sector's entries. If a long name entry is found, the name is build adding all long dir entries until finding the short dir they belong to. When this happens, the long name as well the file's cluster start number are saved in a list of entries.

Finding a DIR\_Name field of 0x00 stops the subroutine as it indicates that all the following entries are empty.

Finding a DIR\_Name field of 0xE5 indicates that the corresponding entry is empty and has no valid data.

After all entries from the retrieved sector have been checked, the program retrieves the data of the next sector in the SD card. If there are no more sectors left to check in that cluster, the program retrieves the corresponding FAT entry obtaining the directory's next cluster and repeating the sector check process. If the value retrieved from the FAT table is greater than 0x0FFFFFFF7 means that there are no more directory clusters to check, ending the routine and returning the saves elements.

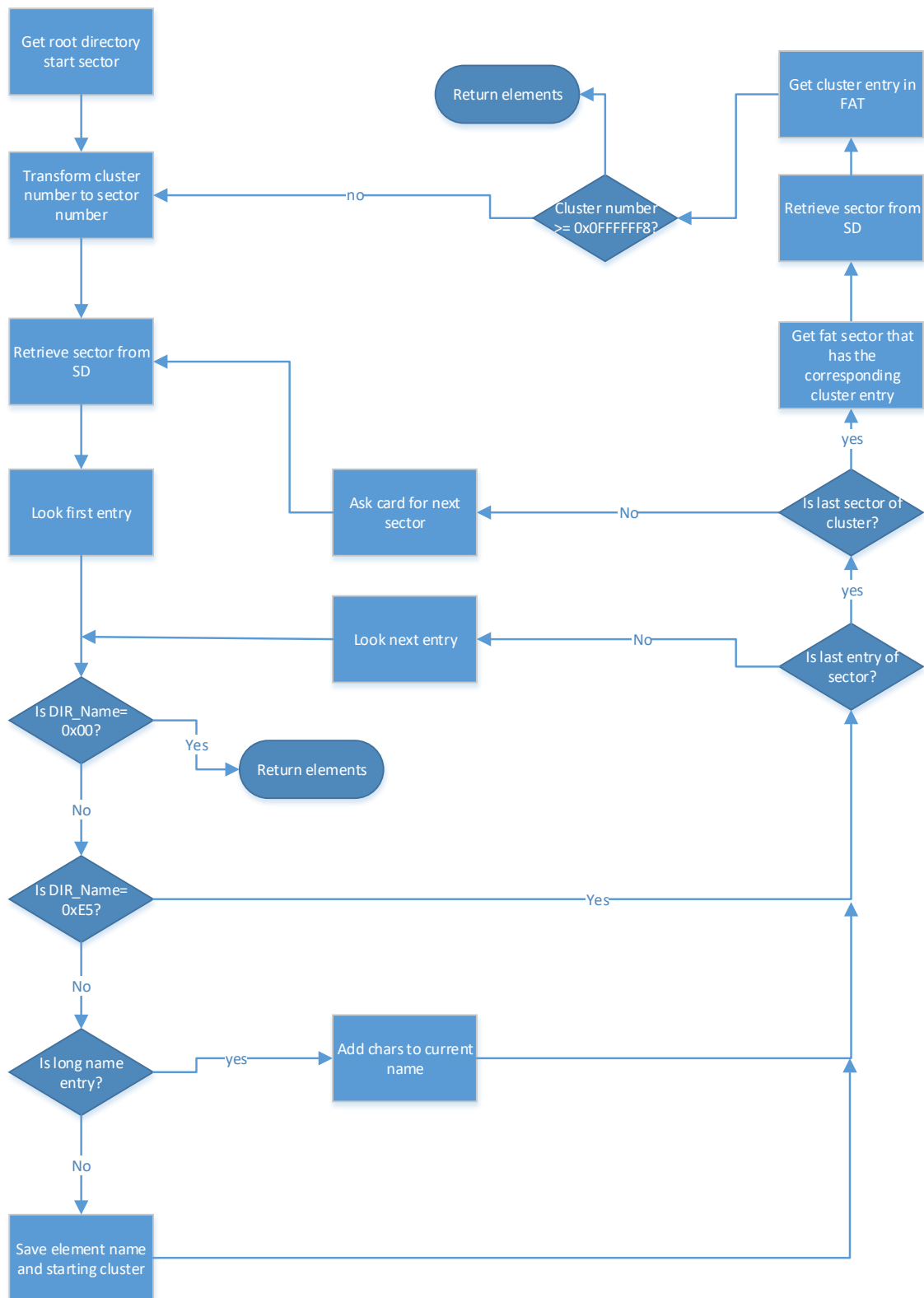


Image 6. 3 flow chart for SD card retrieval

## 6.3 Game example

To check that the platform works perfectly and to ensure that other teams could develop games for G-Engine a simple game was developed as a demonstration.

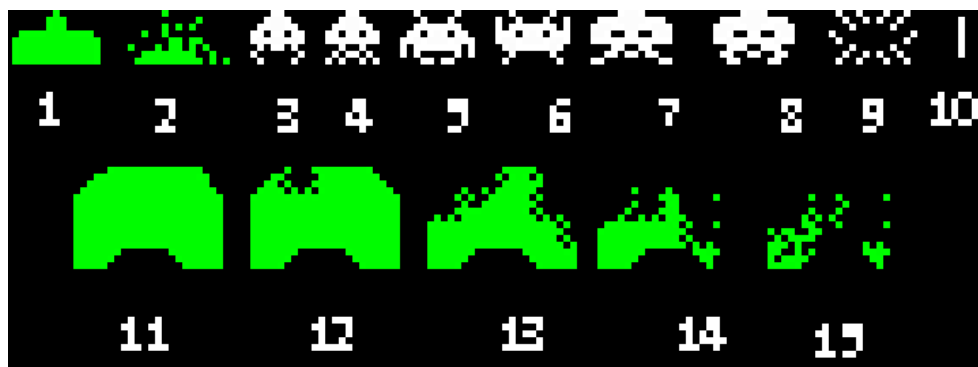
The game of choice was the classic from arcades SPACE INVADERS.

In this game, the player needs to destroy all the alien ships before they land. To do it, the player controls a human ship that can shoot missiles one at a time; until the previous missile has been destroyed the player can't shoot another one. On their side alien ships are also loaded with weapons. The ships on the bottom row will randomly shoot to the earth trying to destroy the player. For this reason, earth has built four shields that can protect the player. After receiving 5 shots the shield will be destroyed, this counts alien and player shots so be careful.

After all alien ships have been destroyed another assault will start. Let's see who can survive longer with the three ships with which everybody starts.

The images used will be the following and can be seen in image 6.7:

1. Player ship
2. Player ship destroyed
3. Alien 1 mode A
4. Alien 1 mode B
5. Alien 2 mode A
6. Alien 2 mode B
7. Alien 3 mode A
8. Alien 3 mode B
9. Alien explosion
10. Missile
11. Shield 100% capacity
12. Shield 80% capacity
13. Shield 60% capacity
14. Shield 40% capacity
15. Shield 20% capacity



*Image 6. 4 sprites images*



### **LCD display:**

The screen will be divided into two areas: the game area (area where the action takes place) and the HUD (Heads Up Display) where status information about the game is displayed, like the number of remaining lives or the score.



*Image 6. 5 LCD Display. on the top the HUD the rest the game area*

### **HUD:**

Here the number of lifes remaining and the total score can be seen.



### **Game area**

Aliens are distributed into 3 rows according to its type. Aliens in the lower row rewards 10 points if destroyed, in the second row the reward is 20 and, in the third, 30.

Here we also find the shield wall and the player ship. Shields are immobile and can resist up to 5 shots, either from the player or from the aliens.

The player ship can only move laterally (up to screen's edge) and shoot. If destroyed, the player loses one life. If player loses all three lives game is over and can be restarted.



The controls for this game are the following:

- Left and right button. Moves the player ship left or right.
- "A" button. Shots a missile.
- Select button. Pauses the game.
- Start. Restarts after Game Over or triggers the next round after all alien ships are destroyed.

## SPRITES

To manage output image, all visible entities will be represented by a separated sprite. A sprite is defined to have the following fields:

- A bitmap array for pixel data
- Height of the sprite
- Width of the sprite
- X Position of the top left pixel
- Y position of the top left pixel
- Speed in the x axis
- Speed in the Y axis
- A flag that indicates if the sprite is visible and, therefore need to be drawn in the screen and taken into account for collision purposes.
- A flag that indicates if the sprite has been modified since the last screen update and, therefore, needs to be redrawn.
- A field indicating sprite's time to die, this is only used when an alien ship is destroyed. This makes that after sometime it becomes not visible.

## Game phases

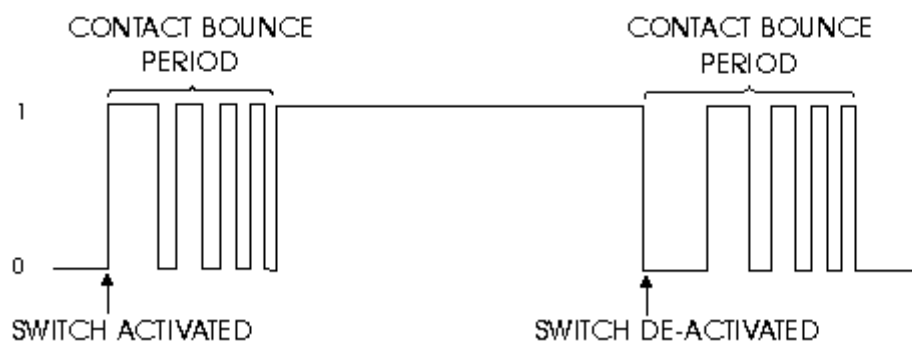
After the component initialization which will configure the TIM2 to run with a period of 500useconds the game has three main phases.

- Gather input commands on the controller
- Calculate game outcome. Update sprite's position using its speed, create a shot after the A button is pressed... all these things are calculated in this phase.
- Using the game outcome, refresh screen.

## Gather input

At first sight this phase seems trivial. It may seem that recover input information from the controller is as easy as reading the IO data in register, but it is not. The behaviour of the push buttons complicates a bit this phase.

After a button is pressed and during a variable amount of time the output value oscillates between high and low. This is a phenomenon is known as bouncing. In image 6.7this phenomenon can be seen.

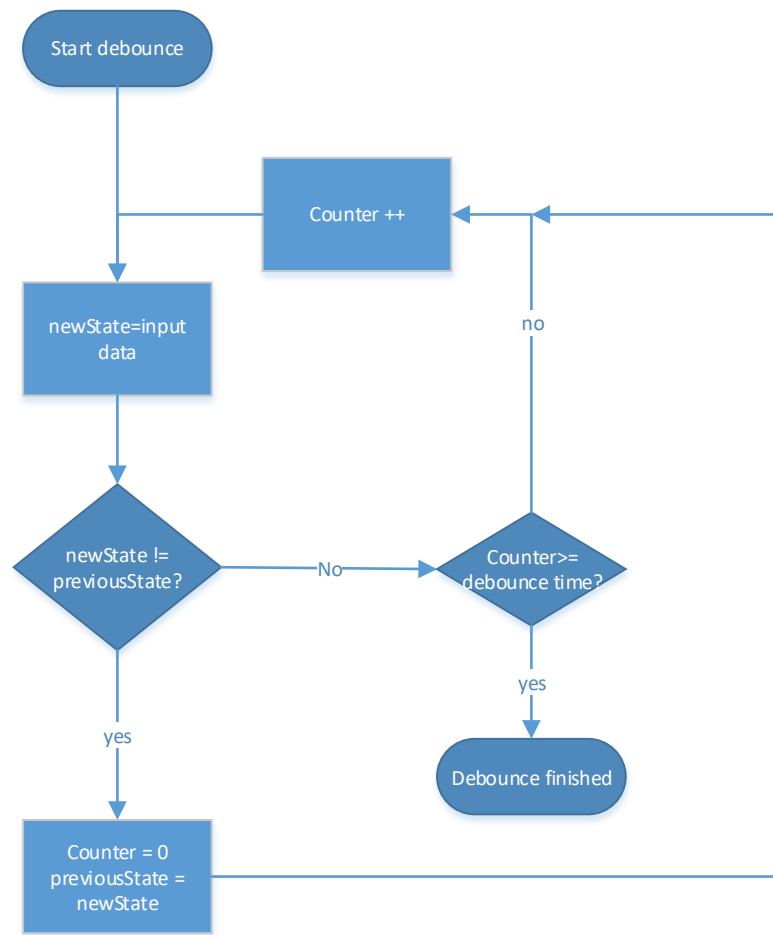


*Figure 1*

*Image 6. 6 button bounce*

This bouncing causes that a single press or release of the button is seen by the processor as several button presses. Because of the way this game is designed the bounce doesn't affect it, but it does affect the SD card reading phase. If we want to select the file or directory just below the current highlighted element, after a single press of the button the highlighted element will be the one several positions below.

As seen in image 6.8 flow chart, to solve this, the program needs to have saved the previous input state. Then, it compares the previous with the new until there is a consistent reading (no change detected for a while). The time necessary to consider a consistent reading is the maximum time between debounces, this can be found by trial and error and in this setup was 4ms.



*Image 6. 7 Debounce flowchart*

### **Calculate game outcome**

To understand this phase is useful to know how the sprites are saved in memory. Sprites are defined as a Struct type, a group of different variable types (Boolean, int, float...). The game code defines the following global sprites so they are consistent within all methods.

- Player sprite
- Player missile sprite
- Array of 4 shield sprites
- Array of 24 alien sprites
- Array of 3 alien missile sprites
- Array of Sprite pointers of all previous sprites.

This way, if the code needs to go through all missiles, but needs to treat different alien missiles from player missiles it can be easily done. If instead is necessary to go through all sprites and treat them the same way, it iterates through the pointer Sprite array.

Note that there is only one sprite for the player missile and it's the same during all the game. To simulate sprite destruction the code changes its visible field to false. The next time the player presses the A button the sprite changes its position to be on top of the player's ship and changes its visible field to true; simulating a new missile is born. The same happens for alien

missiles. In fact that is the reason there can only be one player missile and 3 alien missiles simultaneously on screen.

This phase is divided in the following sub phases:

- Update player speed using controller input data. Left button negative speed right button positive speed.
- Update sprites position using their speed.  $X=x+xspeed$ ,  $y=y+yspeed$ . If new position is out of bounds of screen, position is reverted and speed set to 0, and in case it's a missile destroys it (visible = false).
- Update alien position using the following sequence: lateral movement until an alien collides with the edge of the screen, one move down, lateral on the other direction until an alien collides and down again. Aliens only move one step each 500ms.
- Generate player shot using controller input data. Shoot if A button is pressed and there are no more player missiles on screen(sprite's visible field = false). This sets shot speed to go up until collides with other sprite or exits the screen.
- Randomly generates alien shots. Only the alien closer to the ground of each column is able to shoot. A maximum of 3 alien shots can be on screen simultaneously. This sets shot speed to go down until collides with other sprite or exits the screen.
- Check missile collision with shields, aliens or player. Collision is detected when two sprites have an overlapping region in the screen.
  - o For player missile: the program goes through the alien sprite array and checks if there is collision with the player missile.  
If collision is detected the alien sprite changes its image to the destroyed alien image and sets its time to die so the explosion is only shown for 150ms. The program also adds the corresponding reward points to the score.
  - o For alien missiles. If collision is detected with the player, player image is changed for the dead player image and starts blinking for 3 seconds. After that, the player loses one life and game continues. If player run out of lives, it's game over, the player can press start button to start again.
  - o For both, alien and player missiles. If no other collision is found, the program checks shield collision. If there is collision, the corresponding shield loses one life and changes its image to the next destruction stage. If it loses 5 lives, shield visible is set to false and no longer blocks shots.  
After any collision, or if the missile exits the LCD borders, the missile visible field is set to false.

If any of this sub phases changes the position or the image of a sprite or sets its visible field to false, the needUpdate flag is activated. This will tell the next phase what sprites need to be redrawn. This way the processor doesn't lose time redrawing a sprite that has not changed at all.

### **Refresh screen**

Now that every sprite's information is updated, next step is to show all those changes in the screen.

For this, the program will go through all the sprites pointed in the pointer array, checking the need update flag. If the flag is false, the program skips that sprite and goes for the next one. If true, it can be due 3 things:

- Sprite has moved
- Sprite has changes its image
- Sprite has become not visible
- Sprite has become visible

If the sprite moved, the program compares the new location with the previous one, obtaining the area that was previously occupied by the sprite that became no occupied. Then, sets that area to be background colour (black). After that it proceeds to print the sprite's image in the new location. If the program didn't set that area black, after moving, the sprite would left a ghost image on the screen. The LCD would show a sprite that it's not really there.

If the sprite changed its image is important that the method that changed it also changed sprite's width and height to correspond to the new image. If not, when setting the update window it will not have the dimensions needed to print the image. If that is managed, this step consist simply in setting the update window and draw the sprite.

Lastly, if the update was caused by a change of visibility, the program just draws the sprite in its current position, or draws the background in the area of the sprite, depending is the change was from false to true or from true to false respectively.

There are methods that create update windows outside this phase. These are all related to the HUD, are caused by two things:

- A Score change due an alien's ship destruction.
- A change in the number of lives due to player's ship destruction.

## 7. Prototype cost

### Materials:

Processor board	10.45€
LCD	12.20€
Battery controller	4.50€
Battery	Reutilized, an equivalent one would cost around: 20€
SD card adapter	6.30€
Basic materials: Cables, tin, buttons....	Around 10€
3D printed Case	Around 20€
<b>TOTAL</b>	<b>85€</b>

### Tools & software

ST-Link v2	12€
Keil uvision	Free license
C libraries	Open source
Soldering iron	9€
Cutting tools	8€
Hot glue gun	11€
<b>TOTAL</b>	<b>40€</b>

### Salary

In this project the average was 3-4 hours daily. This leaves us with:

$$1 \text{ worker} \times 3,5 \frac{h}{day} \times 30 \frac{days}{month} \times 4 \text{ months} = 420 h$$

Mean salary for a electronic engineer in Spain: 27.400€/year

Total labor hours in 2018 : 1736h

$$\frac{Price}{h} \text{ of an engineer} = \frac{27400}{1736} = 15.78€/h$$

Total cost of employment = 420x 15.78 = 6.627,60€

The total cost of manufacturing the G-Engine prototype was about 6.752€

## 8. Conclusions

The main aim of this project was to create a functional prototype that was able to load the games from the SD card and execute them, and on this I have been successful. My first idea for this project was to start on the design of the processor board instead of buying one already made. But due to time limitations that idea was discarded after the first meeting with the tutor. It was a wise decision since, in the end, I almost ran out of time. The only regret is that this project ended up being almost exclusively a programming one, but there was little I could do while maintaining the main aim of the project.

On a personal note, one of the objectives was to use this project as an excuse to learn more about microcontrollers and their peripherals. This objective was also widely completed since after all the complications presented by the microcontroller I managed to find good solutions. In addition to do a good research on protocols and interfaces like SD card or FAT32 as I had previously never used them.

Note that as time planning, my job situation made it laborious although ultimately successful. I learned to be better at managing the time of the project.



## 9. Bibliography

- [1] "High-density performance line ARM®-based 32-bit MCU with 256 to 512KB Flash, USB, CAN, 11 timers, 3 ADCs, 13 communication interfaces". Rev 12. November 2015 [online]  
Available: [www.st.com/resource/en/datasheet/stm32f103rc.pdf](http://www.st.com/resource/en/datasheet/stm32f103rc.pdf)
- [2]"STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm® -based 32-bit MCUs" Rev 16. November 2015 [online]-  
Available: [www.st.com/resource/en/reference\\_manual/cd00171190.pdf](http://www.st.com/resource/en/reference_manual/cd00171190.pdf)
- [3]"USART protocol used in the STM32 bootloader" Rev 7. October 2016[online]-  
Available: [www.st.com/resource/en/application\\_note/cd00264342.pdf](http://www.st.com/resource/en/application_note/cd00264342.pdf)
- [4]"STM32 microcontroller system memory boot mode" Rev 31. July 2017[online]-  
Available: [www.st.com/resource/en/application\\_note/cd00167594.pdf](http://www.st.com/resource/en/application_note/cd00167594.pdf)
- [5]"HX8357-B 320RGB x 480 dot, 262K color, with internal GRAM, TFT Mobile Single Chip Driver". Rev. 02 August, 2011 [online]  
Available: <https://www.crystallfontz.com/controllers/Himax/HX8357-B/94/>
- [6]"SanDisk Secure Digital Card. Product Manual" rev 1.9 december 2003 [online]  
Available: [www.convict.lu/pdf/ProdManualSDCardv1.9.pdf](http://www.convict.lu/pdf/ProdManualSDCardv1.9.pdf)
- [7]"Microsoft Extensible Firmware Initiative: FAT32 File System Specification. FAT: General Overview of On-Disk Format" rev1.03, December 2000 [Hardware White paper]  
Available: <https://staff.washington.edu/dittrich/misc/fatgen103.pdf>



Annex 1.4 LCD front view.



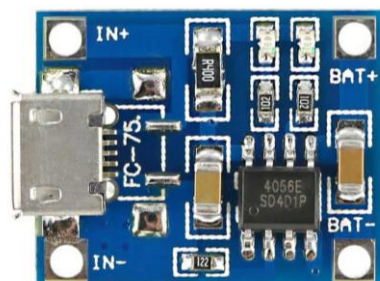
Annex 1.5 LCD back view.



Annex 1.6 Micro SD card to SPI adapter



Annex 1.7 Battery charging PCB



Annex 1.8 Battery 3.7V 5200mAh



Annex 2.1 ST link v2







Annex 3.2 Boot loader USART instructions.

Command <sup>(1)</sup>	Command code	Command description
Get <sup>(2)</sup>	0x00	Gets the version and the allowed commands supported by the current version of the bootloader
Get Version & Read Protection Status <sup>(2)</sup>	0x01	Gets the bootloader version and the Read Protection status of the Flash memory
Get ID <sup>(2)</sup>	0x02	Gets the chip ID
Read Memory <sup>(3)</sup>	0x11	Reads up to 256 bytes of memory starting from an address specified by the application
Go <sup>(3)</sup>	0x21	Jumps to user application code located in the internal Flash memory or in SRAM
Write Memory <sup>(3)</sup>	0x31	Writes up to 256 bytes to the RAM or Flash memory starting from an address specified by the application
Erase <sup>(3)(4)</sup>	0x43	Erases from one to all the Flash memory pages
Extended Erase <sup>(3)(4)</sup>	0x44	Erases from one to all the Flash memory pages using two byte addressing mode (available only for v3.0 usart bootloader versions and above).
Write Protect	0x63	Enables the write protection for some sectors
Write Unprotect	0x73	Disables the write protection for all Flash memory sectors
Readout Protect	0x82	Enables the read protection
Readout Unprotect <sup>(2)</sup>	0x92	Disables the read protection

### Annex 3.3 TIM2\_CR1 register

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

### Annex 3.4 ADC\_SR

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved											STRT	JSTRT	JEOC	EOC	AWD
											rw_w0	rw_w0	rw_w0	rw_w0	rw_w0

Bits 31:5 Reserved, must be kept at reset value.

Bit 4 **STRT**: Regular channel Start flag

This bit is set by hardware when regular channel conversion starts. It is cleared by software.

0: No regular channel conversion started

1: Regular channel conversion has started

Bit 3 **JSTRT**: Injected channel Start flag

This bit is set by hardware when injected channel group conversion starts. It is cleared by software.

0: No injected group conversion started

1: Injected group conversion has started

Bit 2 **JEOC**: Injected channel end of conversion

This bit is set by hardware at the end of all injected group channel conversion. It is cleared by software.

0: Conversion is not complete

1: Conversion complete

Bit 1 **EOC**: End of conversion

This bit is set by hardware at the end of a group channel conversion (regular or injected). It is cleared by software or by reading the ADC\_DR.

0: Conversion is not complete

1: Conversion complete

Bit 0 **AWD**: Analog watchdog flag

This bit is set by hardware when the converted voltage crosses the values programmed in the ADC\_LTR and ADC\_HTR registers. It is cleared by software.

0: No Analog watchdog event occurred

1: Analog watchdog event occurred

## Annex 3.5 ADC\_CR2

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVRE FE	SWSTA RT	JSWST ART	EXTTR IG	EXTSEL[2:0]			Res.
								r/w	r/w	r/w	r/w	r/w	r/w	r/w	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTT RIG	JEXTSEL[2:0]				ALIGN	Reserved	DMA	Reserved				RST CAL	CAL	CONT	ADON
r/w	r/w	r/w	r/w	r/w	Res.		r/w					r/w	r/w	r/w	r/w

### Bit 0 **ADON**: A/D converter ON / OFF

This bit is set and cleared by software. If this bit holds a value of zero and a 1 is written to it then it wakes up the ADC from Power Down state.

Conversion starts when this bit holds a value of 1 and a 1 is written to it. The application should allow a delay of  $t_{STAB}$  between power up and start of conversion. Refer to [Figure 23](#).

0: Disable ADC conversion/calibration and go to power down mode.

1: Enable ADC and to start conversion

*Note: If any other bit in this register apart from ADON is changed at the same time, then conversion is not triggered. This is to prevent triggering an erroneous conversion.*

## Annex 3.6 ADC\_SQR1

Address offset: 0x2C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								L[3:0]				SQ16[4:1]			
								r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]						SQ14[4:0]					SQ13[4:0]			
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:20 **L[3:0]**: Regular channel sequence length

These bits are written by software to define the total number of conversions in the regular channel conversion sequence.

0000: 1 conversion

0001: 2 conversions

.....

1111: 16 conversions

Bits 19:15 **SQ16[4:0]**: 16th conversion in regular sequence

These bits are written by software with the channel number (0..17) assigned as the 16th in the conversion sequence.

Bits 14:10 **SQ15[4:0]**: 15th conversion in regular sequence

Bits 9:5 **SQ14[4:0]**: 14th conversion in regular sequence

Bits 4:0 **SQ13[4:0]**: 13th conversion in regular sequence



### Annex 3.7 ADC\_SQR3

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ6[4:0]					SQ5[4:0]					SQ4[4:1]			
		r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4_0		SQ3[4:0]					SQ2[4:0]					SQ1[4:0]			
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:30 Reserved, must be kept at reset value.

Bits 29:25 **SQ6[4:0]**: 6th conversion in regular sequence

These bits are written by software with the channel number (0..17) assigned as the 6th in the sequence to be converted.

Bits 24:20 **SQ5[4:0]**: 5th conversion in regular sequence

Bits 19:15 **SQ4[4:0]**: 4th conversion in regular sequence

Bits 14:10 **SQ3[4:0]**: 3rd conversion in regular sequence

Bits 9:5 **SQ2[4:0]**: 2nd conversion in regular sequence

Bits 4:0 **SQ1[4:0]**: 1st conversion in regular sequence

### Annex 3.8 ADC\_DR

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADC2DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **ADC2DATA[15:0]**: ADC2 data

In ADC1: In dual mode, these bits contain the regular data of ADC2.

In ADC2 and ADC3: these bits are not used.

Bits 15:0 **DATA[15:0]**: Regular data

These bits are read only. They contain the conversion result from the regular channels.

### Annex 3.9 SPI registers

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0x00	SPI_CR1	Reserved																BIDIMODE	BIDIOE	CRCEN	CRCNEXT	DFE	RXONLY	SSM	SSI	LSBFIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA		
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	SPI_CR2	Reserved																								TXEIE	RXNEIE	ERRIE	Reserved	SSOE	TXDMAEN	RXDMAEN			
	Reset value																									0	0	0		0	0	0	0		
0x08	SPI_SR	Reserved																								BSY	OVR	MODF	CRCERR	UDR	CHSIDE	TXE	RXNE		
	Reset value																									0	0	0	0	0	0	1	0		
0x0C	SPI_DR	Reserved																DR[15:0]																	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x10	SPI_CRCPR	Reserved																CRCPOLY[15:0]																	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1		
0x14	SPI_RXCR	Reserved																RxCRC[15:0]																	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x18	SPI_TXCR	Reserved																TxCRC[15:0]																	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x1C	SPI_I2SCFGR	Reserved																				I2SMOD	I2SE	I2SCFG	PCMSYNC	Reserved	I2SSTD	CKPOL	DATLEN	CHLEN					
	Reset value																					0	0	0	0		0	0	0	0	0	0			
0x20	SPI_I2SPR	Reserved																								MCKOE	ODD	I2SDIV							
	Reset value																									0	0	0	0	0	0	0	0	0	1

Annex 4.1 HX8357-b instruction list 1

(Hex)	Operation Code	DCX	WRX	RDX	D17 ~D8	D7	D6	D5	D4	D3	D2	D1	D0	Function	Display mod Implementation Requirement	
															DM[1:0]	
															00,01,10	11
00	NOP	0	↑	1	-	0	0	0	0	0	0	0	0	No Operation	Y	Y
01	SWRESET	0	↑	1	-	0	0	0	0	0	0	0	1	Software reset	Y	Y
06	RDRED	0	↑	1	-	0	0	0	0	0	1	1	0	Read Red	N	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	R7	R6	R5	R4	R3	R2	R1	R0	-		
07	RDGREN	0	↑	1	-	0	0	0	0	0	1	1	1	Read Green	N	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	G7	G6	G5	G4	G3	G2	G1	G0	-		
08	RDBLUE	0	↑	1	-	0	0	0	0	1	0	0	0	Read Blue	N	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	B7	B6	B5	B4	B3	B2	B1	B0	-		
0A	RDDPM	0	↑	1	-	0	0	0	0	1	0	1	0	Read Display Power Mode	Y	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	D(7:0)							-			
0B	RDDMADCTL	0	↑	1	-	0	0	0	0	1	0	1	1	Read Display MADCTL	Y	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	D(7:0)							-			
0C	RDDCOLMOD	0	↑	1	-	0	0	0	0	1	1	0	0	Read Display Pixel Format	Y	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	x	D6	D5	D4	x	D2	D1	D0	-		
0D	RDDIM	0	↑	1	-	0	0	0	0	1	1	0	1	Read Display Image Mode	Y	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	D(7:0)							-			
0E	RDDSM	0	↑	1	-	0	0	0	0	1	1	1	0	Read Display Signal Mode	Y	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	D(7:0)							-			
0F	RDDSDR	0	↑	1	-	0	0	0	0	1	1	1	1	Read Display Self-Diagnostic Result	Y	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	D(7:0)							-			
10	SLPIN	0	↑	1	-	0	0	0	1	0	0	0	0	Sleep in and charge-pump off	Y	Y
11	SLPOUT	0	↑	1	-	0	0	0	1	0	0	0	1	Sleep out and charge-pump on	Y	Y
12	PTLON	0	↑	1	-	0	0	0	1	0	0	1	0	Partial Mode On	Y	N
13	NORON	0	↑	1	-	0	0	0	1	0	0	1	1	Normal Display Mode On	Y	N

Annex 4.2 HX8357-b instruction list 2

(Hex)	Operation Code	DCX	WRX	RDX	D17~D8	D7	D6	D5	D4	D3	D2	D1	D0	Function	Display mod Implementation Requirement	
															DM[1:0]	
															00,01,10	11
20	INVOFF	0	↑	1	-	0	0	1	0	0	0	0	1	Display Inversion off	Y	Y
21	INVON	0	↑	1	-	0	0	1	0	0	0	1	0	Display Inversion on	Y	Y
28	DISPOFF	0	↑	1	-	0	0	1	0	1	0	0	0	Display off	Y	Y
29	DISPON	0	↑	1	-	0	0	1	0	1	0	0	1	Display on	Y	Y
2A	CASET	0	↑	1	-	0	0	1	0	1	0	1	0	Column Address Set	Y	N
		1	↑	1	-	SC[15:8] (8'b0)								Column address start		
		1	↑	1	-	SC[7:0] (8'b0)								Column address start		
		1	↑	1	-	EC[15:8] (8'b0000_0001)								Column address end		
		1	↑	1	-	EC[7:0] (8'b0011_1111)								Column address end		
2B	PASET	0	↑	1	-	0	0	1	0	1	0	1	1	Row address set	Y	N
		1	↑	1	-	SP[15:8] (8'b0)								Row address start		
		1	↑	1	-	SP[7:0] (8'b0)								Row address start		
		1	↑	1	-	EP[15:8] (8'b0000_0001)								Row address end		
		1	↑	1	-	EP[7:0] (8'b1101_1111)								Row address end		
2C	RAMWR	0	↑	1	-	0	0	1	0	1	1	0	0	Memory write	Y	N
		1	↑	1	-	Write data								-		
2E	RAMRD	0	↑	1	-	0	0	1	0	1	1	1	0	Memory read	Y	N
		1	↑	1	-	-	-	-	-	-	-	-	-	Dummy read		
30	PLTAR	0	↑	1	-	0	0	1	1	0	0	0	0	Partial address set	Y	N
		1	↑	1	-	SR[15:8] (8'b0)								Start row		
		1	↑	1	-	SR[7:0] (8'b0)								Start row		
		1	↑	1	-	ER[15:8] (8'b0000_0001)								End row		
		1	↑	1	-	ER[7:0] (8'b1101_1111)								End row		
33	VSCRDEF	0	↑	1	-	0	0	1	1	0	0	1	1	Vertical Scrolling Definition)	Y	N
		1	↑	1	-	TFA[15:8] (8'b0)								Top Fixed Area		
		1	↑	1	-	TFA[7:0] (8'b0)								Top Fixed Area		
		1	↑	1	-	VSA[15:8] (8'b0000_0001)								Height of the Vertical Scrolling Area		
		1	↑	1	-	VSA[7:0] (8'b1101_1111)								Height of the Vertical Scrolling Area		
		1	↑	1	-	BFA[15:8] (8'b0)								Bottom Fixed Area		
		1	↑	1	-	BFA[7:0] (8'b0)								Bottom Fixed Area		

Annex 4.3 HX8357-b instruction list 3

(Hex)	Operation Code	DCX	WRX	RDX	D17~D8	D7	D6	D5	D4	D3	D2	D1	D0	Function	Display mod Implementation Requirement	
															DM[1:0]	
															00,01,10	11
34	TEOFF	0	↑	1	-	0	0	1	1	0	1	0	0	Tearing Effect line off	Y	N
35	TEON	0	↑	1	-	0	0	1	1	0	1	0	1	Tearing Effect Line ON	Y	N
		0	↑	1	-	-	-	-	-	-	-	-	TEM ODE	-		
36	MADCTL	0	↑	1	-	0	0	1	1	0	1	1	0	Memory Access Control)	Y	Y
		1	↑	1	-	MY (0)	MX (0)	MV (0)	ML (0)	BGR (0)	0	SS (0)	GS (0)	-		
37	VSCRSADD	0	↑	1	-	0	0	1	1	0	1	1	1	Vertical Scrolling Start Address	Y	N
		1	↑	1	-	VSP[15:8] (8'b0)								-		
		1	↑	1	-	VSP[7:0] (8'b0)								-		
38	IDMOFF	0	↑	1	-	0	0	1	1	1	0	0	0	Idle mode off	Y	N
39	IDMON	0	↑	1	-	0	0	1	1	1	0	0	1	Idle mode on	Y	N
3A	COLMOD	0	↑	1	-	0	0	1	1	1	0	1	0	Interface Pixel Format	Y	Y
		1	↑	1	-	0	D[6:4]			0	D[2:0]			-		
3C	RAMWRCON	0	↑	1	-	0	0	1	1	1	1	0	0	Memory write	Y	N
		1	↑	1	-	Write data								-		
3E	RAMRDCON	0	↑	1	-	0	0	1	1	1	1	1	0	Memory read	Y	N
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	Read data								-		
44	TESL	0	↑	1	-	0	1	0	0	0	1	0	0	Set tear scan line	Y	N
		1	↑	1	-	TELINE[15:8] (8'b0)								-		
		1	↑	1	-	TELINE[7:0] (8'b0)								-		
45	GETSL	0	↑	1	-	0	1	0	0	0	1	0	1	Get the current scan line.	Y	N
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	SL[15:8]								-		
		1	1	↑	-	SL[7:0]								-		
A1	Read_DDB_start	0	↑	1	-	1	0	1	0	0	0	0	1	Read the DDB from the provided location.	Y	Y
		1	1	↑	-	-	-	-	-	-	-	-	-	Dummy read		
		1	1	↑	-	ID1								The five bytes always output		
		1	1	↑	-	ID2										
		1	1	↑	-	ID3										
		1	1	↑	-	ID4										
		1	1	↑	-	8'hFF										

Annex 4.4 HX8357-b instruction list 4

(Hex)	Operation Code	DCX	WRX	RDX	D17 ~D8	D7	D6	D5	D4	D3	D2	D1	D0	Function and Default Value
B0	SETEXTC	0	↑	1	-	1	0	1	1	0	0	0	0	Set extended command
		1	↑	1	-	0	0	0	0	0	0	EXTC[1:0]		0x00
B3	SETGRAM	0	↑	1	-	1	0	1	1	0	0	1	1	Set GRAM access and Interface
		1	↑	1	-	0	0	0	0	0	0	0	0	0x02
		1	↑	1	-	0	0	0	0	0	TEI[2:0]			0x00
		1	↑	1	-	0	0	0	0	DENC[3:0]			0x07	
		1	↑	1	-	0	0	EPF[1:0]		0	0	0	DFM	0x00
B4	SETDISPLAY	0	↑	1	-	1	0	1	1	0	1	0	0	Set Display mode and GRAM write mode
		1	↑	1	-	0	0	0	RM	0	0	DM[1:0]		-
BF	GETDEVICEID	0	↑	1	-	1	0	1	1	1	1	1	1	Read Device ID
		1	1	↑	-	MIPI Alliance code								0x01
		1	1	↑	-	MIPI Alliance code								0x62
		1	1	↑	-	Device ID								0x83
		1	1	↑	-	Device ID								0x57
		1	1	↑	-	1	1	1	1	1	1	1	1	0xFF
C0	SETPANEL	0	↑	1	-	1	1	0	0	0	0	0	0	Set Panel Driving
		1	↑	1	-	0	0	0	REV	SM	GS	0	0	0x00
		1	↑	1	-	0	0	NL[5:0]					0x3B-	
		1	↑	1	-	0	SCN[6:0]					0x00		
		1	↑	1	-	0	0	0	NDL	0	PTS[2:0]		0x02	
		1	↑	1	-	0	0	0	PTG	ISC[3:0]			0x11	
C1	SETNORTIME	0	↑	1	-	0	0	0	0	1	0	1	1	Set display timing for Normal mode
		1	↑	1	-	0	0	0	BC0	0	0	DIV0[1:0]		0x10
		1	↑	1	-	0	0	0	RTN0[4:0]				0x17	
		1	↑	1	-	FP0[3:0]				BF0[3:0]				0x24
C2	SETPARTIME	0	↑	1	-	1	1	0	0	0	0	1	0	Set display timing for Partial mode
		1	↑	1	-	0	0	0	BC1	0	0	DIV1[1:0]		0x10
		1	↑	1	-	0	0	0	RTN1[4:0]				0x17	
		1	↑	1	-	FP1[3:0]				BF1[3:0]				0x24
C3	SETIDLTIME	0	↑	1	-	1	1	0	0	0	0	1	1	Set display timing for Idle mode
		1	↑	1	-	0	0	0	BC2	0	0	DIV2[1:0]		0x00
		1	↑	1	-	0	0	0	RTN2[4:0]				0x17	
		1	↑	1	-	FP2[3:0]				BF2[3:0]				0x24
C5	SETOSC	0	↑	1	-	1	1	0	0	0	1	0	1	Set display frame
		1	↑	1	-	0	0	0	0	UADJ[3:0]				0x08
C6	SETRGB	0	↑	1	-	1	1	0	0	0	0	0	0	Set RGB Interface
		1	↑	1	-	SDA EN	0	0	VPL	HPL	0	EPL	DPL	0x02

## Annex 5.1 SD card commands 1

<b>CMD INDEX</b>	<b>SPI Mode</b>	<b>Argument</b>	<b>Resp</b>	<b>Abbreviation</b>	<b>Command Description</b>
CMD0	Yes	None	R1	GO_IDLE_STATE	Resets the SD Card
CMD1	Yes	None	R1	SEND_OP_COND	Activates the card's initialization process.
CMD2	No				
CMD3	No				
CMD4	No				
CMD5	Reserved				
CMD6	Reserved				
CMD7	No				
CMD8	Reserved				
CMD9	Yes	None	R1	SEND_CSD	Asks the selected card to send its card-specific data (CSD).
CMD10	Yes	None	R1	SEND_CID	Asks the selected card to send its card identification (CID).
CMD11	No				
CMD12	Yes	None	R1b	STOP_TRANSMISSION	Forces the card to stop transmission during a multiple block read operation.
CMD13	Yes	None	R2	SEND_STATUS	Asks the selected card to send its status register.
CMD14	No				
CMD15	No				
CMD16	Yes	[31:0] block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read & write). <sup>1</sup>
CMD17	Yes	[31:0] data address	R1	READ_SINGLE_BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command. <sup>2</sup>
CMD18	Yes	[31:0] data address	R1	READ_MULTIPLE_BLOCK	Continuously transfers data blocks from card to host until interrupted by a STOP_TRANSMISSION command.
CMD19	Reserved				
CMD20	No				
CMD21 ... CMD23	Reserved				
CMD24	Yes	[31:0] data address	R1 <sup>3</sup>	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command. <sup>4</sup>
CMD25	Yes	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a stop transmission token is sent (instead of 'start block').

## Annex 5.2 SD card commands 2

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD26	No				
CMD27	Yes	None	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28 <sup>1</sup>	Yes	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card specific data (WVP_GRP_SIZE).
CMD29 <sup>4</sup>	Yes	[31:0] data address	R1b	CLR_WRITE_PROT	If the card has write protection features, this command clears the write protection bit of the addressed group.
CMD30	Yes	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card has write protection features, this command asks the card to send the status of the write protection bits. <sup>2</sup>
CMD31	Reserved				
CMD32	Yes	[31:0] data address	R1	ERASE_WVR_BLK_START_ADDR	Sets the address of the first write block to be erased.
CMD33	Yes	[31:0] data address	R1	ERASE_WVR_BLK_END_ADDR	Sets the address of the last write block in a continuous range to be erased.
CMD34 .... CMD37	Reserved				
CMD38	Yes	[31:0] don't care*	R1b	ERASE	Erases all previously selected write blocks.
CMD39	No				
CMD40	No				
CMD41 ... CMD54	Reserved				
CMD55	Yes	[31:0] stuff bits	R1	APP_CMD	Notifies the card that the next command is an application specific command rather than a standard command.
CMD56	Yes	[31:0] stuff bits [0]: RD/WVR. <sup>3</sup>	R1	GEN_CMD	Used either to transfer a Data Block to the card or to get a Data Block from the card for general purpose/application specific commands. The size of the Data Block is defined with SET_BLOCK_LEN command.
CMD57	Reserved				
CMD58	Yes	None	R3	READ_OCR	Reads the OCR register of a card.
CMD59	Yes	[31:1] don't care* [0:0] CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on, a '0' will turn it off.
CMD60-63	No				

\* The bit places must be filled but the values are irrelevant.



### Annex 5.3 Application SD card commands

<b>CMD INDEX</b>	<b>SPI Mode</b>	<b>Argument</b>	<b>Resp</b>	<b>Abbreviation</b>	<b>Command Description</b>
ACMD6	No				
ACMD13	Yes	[31:0] stuff bits	R2	SD_STATUS	Send the SD Card status. The status fields are given in Table 4-21
ACMD17	Reserved				
ACMD18	Yes	–	--	–	Reserved for SD security applications <sup>1</sup>
ACMD19 to ACMD21	Reserved				
ACMD22	Yes	[31:0] stuff bits	R1	SEND_NUM_WR_BLOCKS	Send the numbers of the well-written (without errors) blocks. Responds with 32bit+CRC data block.
ACMD23	Yes	[31:23] stuff bits [22:0] Number of blocks	R1	SET_WR_BLK_ERASE_COUNT	Set the number of write blocks to be pre-erased before writing (to be used for faster Multiple Block WR command). "1"=default (one wr block) (2).
ACMD24	Reserved				
ACMD25	Yes	–	--	–	Reserved for SD security applications <sup>1</sup>
ACMD26	Yes	–	--	–	Reserved for SD security applications <sup>1</sup>
ACMD38	Yes	–	--	–	Reserved for SD security applications <sup>1</sup>
ACMD39 to ACMD40	Reserved				
ACMD41	Yes	None	R1	SEND_OP_COND	Activates the card's initialization process.
ACMD42	Yes	[31:1] stuff bits [0]set_cd	R1	SET_CLR_CARD_DETECT	Connect[1]/Disconnect[0] the 50 KOhm pull-up resistor on CD/DAT3 (pin 1) of the card. The pull-up may be used for card detection.
ACMD43 ... ACMD49	Yes	–	--	–	Reserved for SD security applications. <sup>1</sup>
ACMD51	Yes	[31:0] stuff bits	R1	SEND_SCR	Reads the SD Configuration Register (SCR).

## Annex 5.4 CSD structure

Name	Field	Width	Cell Type	CSD-Slice	CSD Value	CSD Code
CSD structure	CSD_STRUCTURE	2	R	[127:126]	1.0	00b
Reserved	-	6	R	[125:120]	-	000000b
data read access-time-1	TAAC Binary MLC	8 8	R R	[119:112] [119:112]	1.5msec 10msec	00100110b 00001111b
data read access-time-2 in CLK cycles (NSA C*100)	NSAC	8	R	[111:104]	0	00000000b
max. data transfer rate	TRAN_SPEED	8	R	[103:96]	25MHz	00110010b
card command classes	CCC	12	R	[95:84]	All (incl. WP, Lock/unlock)	1F5h
max. read data block length	READ_BL_LEN	4	R	[83:80]	512byte	1001b
partial blocks for read allowed	READ_BL_PARTIAL	1	R	[79:79]	Yes	1b
write block misalignment	WRITE_BLK_MISALIGN	1	R	[78:78]	No	0b
read block misalignment	READ_BLK_MISALIGN	1	R	[77:77]	No	0b
DSR implemented	DSR_IMP	1	R	[76:76]	No	0b
Reserved	-	2	R	[75:74]	-	00b
device size	C_SIZE	12	R	[73:62]	SD128=3843 SD064=3807 SD032=1867 SD016=899 SD008=831	F03h EDFh 74Bh 383h 33Fh
max. read current @VDD min	VDD_R_CURR_MIN	3	R	[61:59]	100mA	111b
max. read current @VDD max	VDD_R_CURR_MAX	3	R	[58:56]	80mA	110b
max. write current @VDD min	VDD_W_CURR_MIN	3	R	[55:53]	100mA	111b
max. write current @VDD max	VDD_W_CURR_MAX	3	R	[52:50]	80mA	110b
device size multiplier	C_SIZE_MULT	3	R	[49:47]	SD128=64 SD064=32 SD032=32 SD016=32 SD008=16	100b 011b 011b 011b 010b
erase single block enable	ERASE_BLK_EN	1	R	[46:46]	Yes	1b
erase sector size	SECTOR_SIZE	7	R	[45:39]	32blocks	0011111b
write protect group size	WP_GRP_SIZE	7	R	[38:32]	128sectors	1111111b
write protect group enable	WP_GRP_ENABLE	1	R	[31:31]	Yes	1b
Reserved for MultiMediaCard compatibility		2	R	[30:29]	-	00b
write speed factor Binary MLC	R2W_FACTOR R2W_FACTOR	3 3	R R	[1:16] [1:4]	X16 X4	100b 010b
max. write data block length	WRITE_BL_LEN	4	R	[25:22]	512Byte	1001b
partial blocks for write allowed	WRITE_BL_PARTIAL	1	R	[21:21]	No	0
Reserved	-	5	R	[20:16]	-	00000b
File format group	FILE_FORMAT_GRP	1	R/W(1)	[15:15]	0	0b
copy flag (OTP)	COPY	1	R/W(1)	[14:14]	Not Original	1b
permanent write protection	PERM_WRITE_PROTECT	1	R/W(1)	[13:13]	Not Protected	0b
temporary write protection	TMP_WRITE_PROTECT	1	R/W	[12:12]	Not Protected	0b
File format	FILE_FORMAT	2	R/W(1)	[11:10]	HD w/partition	00b
Reserved	-	2	R/W	[9:8]	-	00b
CRC	CRC	7	R/W	[7:1]	-	CRC7
not used, always '1'	-	1	-	[0:0]	-	1b

## Annex 5.5 BPB in FAT32

### Boot Sector and BPB Structure

Name	Offset (byte)	Size (bytes)	Description
BS_jmpBoot	0	3	<p>Jump instruction to boot code. This field has two allowed forms:  <b>jmpBoot[0] = 0xEB, jmpBoot[1] = 0x??, jmpBoot[2] = 0x90</b>  and  <b>jmpBoot[0] = 0xE9, jmpBoot[1] = 0x??, jmpBoot[2] = 0x??</b></p> <p><b>0x??</b> indicates that any 8-bit value is allowed in that byte. What this forms is a three-byte Intel x86 unconditional branch (jump) instruction that jumps to the start of the operating system bootstrap code. This code typically occupies the rest of sector 0 of the volume following the BPB and possibly other sectors. Either of these forms is acceptable. <b>JmpBoot[0] = 0xEB</b> is the more frequently used format.</p>
BS_OEMName	3	8	<p>“MSWIN4.1” There are many misconceptions about this field. It is only a name string. Microsoft operating systems don’t pay any attention to this field. Some FAT drivers do. This is the reason that the indicated string, “MSWIN4.1”, is the recommended setting, because it is the setting least likely to cause compatibility problems. If you want to put something else in here, that is your option, but the result may be that some FAT drivers might not recognize the volume. Typically this is some indication of what system formatted the volume.</p>
BPB_BytsPerSec	11	2	<p>Count of bytes per sector. This value may take on only the following values: 512, 1024, 2048 or 4096. If maximum compatibility with old implementations is desired, only the value 512 should be used. There is a lot of FAT code in the world that is basically “hard wired” to 512 bytes per sector and doesn’t bother to check this field to make sure it is 512. Microsoft operating systems will properly support 1024, 2048, and 4096.</p> <p><b>Note:</b> Do not misinterpret these statements about maximum compatibility. If the media being recorded has a physical sector size N, you must use N and this must still be less than or equal to 4096. Maximum compatibility is achieved by only using media with specific sector sizes.</p>
BPB_SecPerClus	13	1	<p>Number of sectors per allocation unit. This value must be a power of 2 that is greater than 0. The legal values are 1, 2, 4, 8, 16, 32, 64, and 128. Note however, that a value should never be used that results in a “bytes per cluster” value (BPB_BytsPerSec * BPB_SecPerClus) greater than 32K (32 * 1024). There is a misconception that values greater than this are OK. Values that cause a cluster size greater than 32K bytes do not work properly; do not try to define one. Some versions of some systems allow 64K bytes per cluster value. Many application setup programs will not work correctly on such a FAT volume.</p>
BPB_RsvdSecCnt	14	2	<p>Number of reserved sectors in the Reserved region of the volume starting at the first sector of the volume. This field must not be 0. For FAT12 and FAT16 volumes, this value should never be anything other than 1. For FAT32 volumes, this value is typically 32. There is a lot of FAT code in the world “hard wired” to 1 reserved sector for FAT12 and FAT16 volumes and that doesn’t bother to check this field to make sure it is 1. Microsoft operating systems will properly support any non-zero value in this field.</p>

BPB_NumFATs	16	1	<p>The count of FAT data structures on the volume. This field should always contain the value 2 for any FAT volume of any type. Although any value greater than or equal to 1 is perfectly valid, many software programs and a few operating systems' FAT file system drivers may not function properly if the value is something other than 2. All Microsoft file system drivers will support a value other than 2, but it is still highly recommended that no value other than 2 be used in this field.</p> <p>The reason the standard value for this field is 2 is to provide redundancy for the FAT data structure so that if a sector goes bad in one of the FATs, that data is not lost because it is duplicated in the other FAT. On non-disk-based media, such as FLASH memory cards, where such redundancy is a useless feature, a value of 1 may be used to save the space that a second copy of the FAT uses, but some FAT file system drivers might not recognize such a volume properly.</p>
BPB_RootEntCnt	17	2	For FAT12 and FAT16 volumes, this field contains the count of 32-byte directory entries in the root directory. For FAT32 volumes, this field must be set to 0. For FAT12 and FAT16 volumes, this value should always specify a count that when multiplied by 32 results in an even multiple of BPB_BytsPerSec. For maximum compatibility, FAT16 volumes should use the value 512.
BPB_TotSec16	19	2	This field is the old 16-bit total count of sectors on the volume. This count includes the count of all sectors in all four regions of the volume. This field can be 0; if it is 0, then BPB_TotSec32 must be non-zero. For FAT32 volumes, this field must be 0. For FAT12 and FAT16 volumes, this field contains the sector count, and BPB_TotSec32 is 0 if the total sector count "fits" (is less than 0x10000).
BPB_Media	21	1	0xF8 is the standard value for "fixed" (non-removable) media. For removable media, 0xF0 is frequently used. The legal values for this field are 0xF0, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, and 0xFF. The only other important point is that whatever value is put in here must also be put in the low byte of the FAT[0] entry. This dates back to the old MS-DOS 1.x media determination noted earlier and is no longer usually used for anything.
BPB_FATSz16	22	2	This field is the FAT12/FAT16 16-bit count of sectors occupied by ONE FAT. On FAT32 volumes this field must be 0, and BPB_FATSz32 contains the FAT size count.
BPB_SecPerTrk	24	2	Sectors per track for interrupt 0x13. This field is only relevant for media that have a geometry (volume is broken down into tracks by multiple heads and cylinders) and are visible on interrupt 0x13. This field contains the "sectors per track" geometry value.
BPB_NumHeads	26	2	Number of heads for interrupt 0x13. This field is relevant as discussed earlier for BPB_SecPerTrk. This field contains the one based "count of heads". For example, on a 1.44 MB 3.5-inch floppy drive this value is 2.
BPB_HiddSec	28	4	Count of hidden sectors preceding the partition that contains this FAT volume. This field is generally only relevant for media visible on interrupt 0x13. This field should always be zero on media that are not partitioned. Exactly what value is appropriate is operating system specific.
BPB_TotSec32	32	4	This field is the new 32-bit total count of sectors on the volume. This count includes the count of all sectors in all four regions of the volume. This field can be 0; if it is 0, then BPB_TotSec16 must be non-zero. For FAT32 volumes, this field must be non-zero. For FAT12/FAT16 volumes, this field contains the sector count if BPB_TotSec16 is 0 (count is greater than or equal to 0x10000).

Name	Offset (byte)	Size (bytes)	Description
BPB_FATsSz32	36	4	This field is only defined for FAT32 media and does not exist on FAT12 and FAT16 media. This field is the FAT32 32-bit count of sectors occupied by ONE FAT. BPB_FATsSz16 must be 0.
BPB_ExtFlags	40	2	This field is only defined for FAT32 media and does not exist on FAT12 and FAT16 media. Bits 0-3 -- Zero-based number of active FAT. Only valid if mirroring is disabled. Bits 4-6 -- Reserved. Bit 7 -- 0 means the FAT is mirrored at runtime into all FATs. -- 1 means only one FAT is active; it is the one referenced in bits 0-3. Bits 8-15 -- Reserved.
BPB_FSVer	42	2	This field is only defined for FAT32 media and does not exist on FAT12 and FAT16 media. High byte is major revision number. Low byte is minor revision number. This is the version number of the FAT32 volume. This supports the ability to extend the FAT32 media type in the future without worrying about old FAT32 drivers mounting the volume. This document defines the version to 0:0. If this field is non-zero, back-level Windows versions will not mount the volume. <b>NOTE:</b> Disk utilities should respect this field and not operate on volumes with a higher major or minor version number than that for which they were designed. FAT32 file system drivers must check this field and not mount the volume if it does not contain a version number that was defined at the time the driver was written.
BPB_RootClus	44	4	This field is only defined for FAT32 media and does not exist on FAT12 and FAT16 media. This is set to the cluster number of the first cluster of the root directory, usually 2 but not required to be 2. <b>NOTE:</b> Disk utilities that change the location of the root directory should make every effort to place the first cluster of the root directory in the first non-bad cluster on the drive (i.e., in cluster 2, unless it's marked bad). This is specified so that disk repair utilities can easily find the root directory if this field accidentally gets zeroed.
BPB_FSInfo	48	2	This field is only defined for FAT32 media and does not exist on FAT12 and FAT16 media. Sector number of FSINFO structure in the reserved area of the FAT32 volume. Usually 1. <b>NOTE:</b> There will be a copy of the FSINFO structure in BackupBoot, but only the copy pointed to by this field will be kept up to date (i.e., both the primary and backup boot record will point to the same FSINFO sector).
BPB_BkBootSec	50	2	This field is only defined for FAT32 media and does not exist on FAT12 and FAT16 media. If non-zero, indicates the sector number in the reserved area of the volume of a copy of the boot record. Usually 6. No value other than 6 is recommended.
BPB_Reserved	52	12	This field is only defined for FAT32 media and does not exist on FAT12 and FAT16 media. Reserved for future expansion. Code that formats FAT32 volumes should always set all of the bytes of this field to 0.
BS_DrvNum	64	1	This field has the same definition as it does for FAT12 and FAT16 media. The only difference for FAT32 media is that the field is at a different offset in the boot sector.
BS_Reserved1	65	1	This field has the same definition as it does for FAT12 and FAT16 media. The only difference for FAT32 media is that the field is at a different offset in the boot sector.

BS_BootSig	66	1	This field has the same definition as it does for FAT12 and FAT16 media. The only difference for FAT32 media is that the field is at a different offset in the boot sector.
BS_VolID	67	4	This field has the same definition as it does for FAT12 and FAT16 media. The only difference for FAT32 media is that the field is at a different offset in the boot sector.
BS_VolLab	71	11	This field has the same definition as it does for FAT12 and FAT16 media. The only difference for FAT32 media is that the field is at a different offset in the boot sector.
BS_FilSysType	82	8	Always set to the string " <b>FAT32</b> ". Please see the note for this field in the FAT12/FAT16 section earlier. This field has nothing to do with FAT type determination.

## Annex 5.6 directory Entry byte structure

### FAT 32 Byte Directory Entry Structure

Name	Offset (byte)	Size (bytes)	Description
DIR_Name	0	11	Short name.
DIR_Attr	11	1	File attributes: ATTR_READ_ONLY 0x01 ATTR_HIDDEN 0x02 ATTR_SYSTEM 0x04 ATTR_VOLUME_ID 0x08 ATTR_DIRECTORY 0x10 ATTR_ARCHIVE 0x20 ATTR_LONG_NAME ATTR_READ_ONLY   ATTR_HIDDEN   ATTR_SYSTEM   ATTR_VOLUME_ID  The upper two bits of the attribute byte are reserved and should always be set to 0 when a file is created and never modified or looked at after that.
DIR_NTRes	12	1	Reserved for use by Windows NT. Set value to 0 when a file is created and never modify or look at it after that.
DIR_CrtTimeTenth	13	1	Millisecond stamp at file creation time. This field actually contains a count of tenths of a second. The granularity of the seconds part of DIR_CrtTime is 2 seconds so this field is a count of tenths of a second and its valid value range is 0-199 inclusive.
DIR_CrtTime	14	2	Time file was created.
DIR_CrtDate	16	2	Date file was created.
DIR_LstAccDate	18	2	Last access date. Note that there is no last access time, only a date. This is the date of last read or write. In the case of a write, this should be set to the same date as DIR_WrtDate.
DIR_FstClusHI	20	2	High word of this entry's first cluster number (always 0 for a FAT12 or FAT16 volume).
DIR_WrtTime	22	2	Time of last write. Note that file creation is considered a write.
DIR_WrtDate	24	2	Date of last write. Note that file creation is considered a write.
DIR_FstClusLO	26	2	Low word of this entry's first cluster number.
DIR_FileSize	28	4	32-bit DWORD holding this file's size in bytes.

## Annex 5.7 Long Name directory entry byte structure

### **FAT Long Directory Entry Structure**

Name	Offset (byte)	Size (bytes)	Description
LDIR_Ord	0	1	The order of this entry in the sequence of long dir entries associated with the short dir entry at the end of the long dir set.  If masked with 0x40 (LAST_LONG_ENTRY), this indicates the entry is the last long dir entry in a set of long dir entries. All valid sets of long dir entries must begin with an entry having this mask.
LDIR_Name1	1	10	Characters 1-5 of the long-name sub-component in this dir entry.
LDIR_Attr	11	1	Attributes - must be ATTR_LONG_NAME
LDIR_Type	12	1	If zero, indicates a directory entry that is a sub-component of a long name. NOTE: Other values reserved for future extensions.  Non-zero implies other dirent types.
LDIR_Chksum	13	1	Checksum of name in the short dir entry at the end of the long dir set.
LDIR_Name2	14	12	Characters 6-11 of the long-name sub-component in this dir entry.
LDIR_FstClusLO	26	2	Must be ZERO. This is an artifact of the FAT "first cluster" and must be zero for compatibility with existing disk utilities. It's meaningless in the context of a long dir entry.
LDIR_Name3	28	4	Characters 12-13 of the long-name sub-component in this dir entry.

## Annex 6.1 Main:

```
#include "delay.h"
#include "sys.h"
#include "lcd.h"
#include "spi.h"
#include "flash.h"
#include "mmc_sd.h"
#include "FAT32.h"
#include "integer.h"
#include "fontupd.h"
#include "stdio.h"
#include <stdbool.h>
#include "Timer2.h"
#include "LoadMenu.h"
#include "Controller.h"
#include "game.h"
#include "adc.h"
#include "stm32f10x.h"

DWORD addres=0x8010000; //address where the game code will be loaded.

bool SDInit=false;
int main(void)
{

    void (*JumpToGame)(void) = (void (*)(void))addres; //create a
dummy function that when called jumps to game code

    DWORD tim=millis();
    FATDir dir;
    dir.elementCount=0;

    SystemInit();
    delay_init(72);
    NVIC_Configuration();
    SPI_Flash_Init();
    SPIx_SetSpeed(SPI_BaudRatePrescaler_256);
    Timer2_init();
    ControllerInit(GPIOE,RCC_APB2Periph_GPIOE);
    Adc_Init();
    LCD_Init();
    LoadMenuInit();
    checkBattery(); // initialize all peripherals used

    while(SDInit==false){ //don't continue unless sd
card has been initialized,
        SDInit=!SD_Initialize();
        if(millis()-tim>1500){
            writeError((u8*)"SD Card could not initialize.");//if
it takes too long show error message
        }
        initializeFat32();
    }
    writeError(0); //delete error message if needed

    getRootDirectory(&dir);
    loadContent(&dir); //get the root directory from the sd card
and display the contents in the LCD
```



```

while(1){
    getButtonState(); //get the buttons state
    if(getStateChange()){
        if(getUp()){//navigate though the directory
            goUp();
        }
        if(getDown()){
            goDown();
        }
        if(getActionA()){
            writeError(0); //delete error message if needed
            FatElement element=open(); //stores the element
selected,
            if(element.Attr&0x10){ //if its a directory go to it
                getDirectoryContent(element.FstClus,&dir);
                loadContent(&dir);
            }
            else if(extensionCheck(element)){//if its a file
checks that is a HEX file
                LoadGame(element.FstClus,address); // copies the
code from the HEX file into memory
                JumpToGame();//jumps to the defined address
            }
            else{// if not HEX show error
                writeError((u8*)"Error, not a game File.");
            }
        }
    }
}

bool extensionCheck(FatElement element){
    int i;
    u8 check[3];
    u8* name=element.name;
    while(*name!=0&&*name!=0xff){// save the 3 last characters of the
name
        for(i=0;i<3;i++){
            check[i]=*name;
            name++;
        }
        if(check[0]!='h') return false;//check wether or not the file
extension is .HEX
        if(check[1]!='e') return false;
        if(check[2]!='x') return false;
        return true;
    }
}

```

## Annex 6.2 Fat32.h and FAT32.c

```
#ifndef FAT32
#define FAT32

#include "integer.h"    /* Basic integer types */
#include "stdbool.h"

/* Multi-byte word access macros */
//gets an address in a big endian memory and returns a WORD or DWORD
in big endian

/* Use byte-by-byte access to the FAT structure */
#define LD_WORD(ptr) \
(WORD) (( (WORD) * ( (BYTE*) (ptr) + 1) << 8) | (WORD) * (BYTE*) (ptr) )
#define LD_DWORD(ptr) \
(DWORD) (( (DWORD) * ( (BYTE*) (ptr) + 3) << 24) | ( (DWORD) * ( (BYTE*) (ptr) + 2) << 16) | \
( (WORD) * ( (BYTE*) (ptr) + 1) << 8) | * (BYTE*) (ptr) )
#define ST_WORD(ptr, val) \
* ( (BYTE*) (ptr) ) = (BYTE) (val); \
* ( (BYTE*) (ptr) + 1) = (BYTE) ( (WORD) (val) >> 8)
#define ST_DWORD(ptr, val) \
* ( (BYTE*) (ptr) ) = (BYTE) (val); \
* ( (BYTE*) (ptr) + 1) = (BYTE) ( (WORD) (val) >> 8); \
* ( (BYTE*) (ptr) + 2) = (BYTE) ( (DWORD) (val) >> 16); \
* ( (BYTE*) (ptr) + 3) = (BYTE) ( (DWORD) (val) >> 24)

/* FatFs refers the members in the FAT structures as byte array
instead of
/ structure member because the structure is not binary compatible
between
/ different platforms */

#define BS_jmpBoot 0 /* Jump instruction (3) */
#define BS_OEMName 3 /* OEM name (8) */
#define BPB_BytsPerSec 11 /* Sector size [byte] (2) */
#define BPB_SecPerClus 13 /* Cluster size [sector] (1) */
#define BPB_RsvdSecCnt 14 /* Size of reserved area [sector] (2)
*/
#define BPB_NumFATs 16 /* Number of FAT copies (1) */
#define BPB_RootEntCnt 17 /* Number of root dir entries for
FAT12/16 (2) */
#define BPB_TotSec16 19 /* Volume size [sector] (2) */
#define BPB_Media 21 /* Media descriptor (1) */
#define BPB_FATSz16 22 /* FAT size [sector] (2) */
#define BPB_SecPerTrk 24 /* Track size [sector] (2) */
#define BPB_NumHeads 26 /* Number of heads (2) */
#define BPB_HiddSec 28 /* Number of special hidden sectors
(4) */
#define BPB_TotSec32 32 /* Volume size [sector] (4) */
#define BS_DrvNum 36 /* Physical drive number (2) */
#define BS_BootSig 38 /* Extended boot signature (1) */
#define BS_VolID 39 /* Volume serial number (4) */
#define BS_VolLab 43 /* Volume label (8) */
#define BS_FilSysType 54 /* File system type (1) */
#define BPB_FATSz32 36 /* FAT size [sector] (4) */
#define BPB_ExtFlags 40 /* Extended flags (2) */
#define BPB_FSVer 42 /* File system version (2) */
#define BPB_RootClus 44 /* Root dir first cluster (4) */
#define BPB_FSInfo 48 /* Offset of FSInfo sector (2) */
```

```

#define BPB_BkBootSec      50  /* Offset of backup boot sectot (2) */
#define BS_DrvNum32        64  /* Physical drive number (2) */
#define BS_BootSig32       66  /* Extended boot signature (1) */
#define BS_VolID32         67  /* Volume serial number (4) */
#define BS_VolLab32        71  /* Volume label (8) */
#define BS_FilSysType32    82  /* File system type (1) */
#define FSI_LeadSig        0   /* FSI: Leading signature (4) */
#define FSI_StrucSig       484 /* FSI: Structure signature (4) */
#define FSI_Free_Count     488 /* FSI: Number of free clusters (4) */
#define FSI_Nxt_Free      492 /* FSI: Last allocated cluster (4) */
#define MBR_Table         446 /* MBR: Partition table offset (2) */
#define SZ_PTE            16  /* MBR: Size of a partition table
entry */
#define BS_55AA           510 /* Boot sector signature (2) */

#define DIR_Name           0   /* Short file name (11) */
#define DIR_Attr           11  /* Attribute (1) */
#define DIR_NTres          12  /* NT flag (1) */
#define DIR_CrtTime        14  /* Created time (2) */
#define DIR_CrtDate        16  /* Created date (2) */
#define DIR_FstClusHI      20  /* Higher 16-bit of first cluster (2)
*/
#define DIR_WrtTime        22  /* Modified time (2) */
#define DIR_WrtDate        24  /* Modified date (2) */
#define DIR_FstClusLO      26  /* Lower 16-bit of first cluster (2)
*/
#define DIR_FileSize       28  /* File size (4) */
#define LDIR_Ord            0   /* LFN entry order and LLE flag (1) */
#define LDIR_Attr          11  /* LFN attribute (1) */
#define LDIR_Type          12  /* LFN type (1) */
#define LDIR_Chksum        13  /* Sum of corresponding SFN entry */
#define LDIR_FstClusLO     26  /* Filled by zero (0) */
#define SZ_DIR             32  /* Size of a directory entry */
#define LLE                0x40 /* Last long entry flag in
LDIR_Ord */
#define DDE                0xE5 /* Deleted directory enrty mark in
DIR_Name[0] */
#define NDDE              0x05 /* Replacement of a character
collides with DDE */

/* Usefull data structures used */

typedef struct{
    WORD  BytsPerSect;
    BYTE  SecPerClus;
    WORD  RsvdSecCnt;
    BYTE  NumFATs;
    DWORD FATSz32;
    DWORD RootClus;
}BPB;

typedef BYTE SECTOR[512];

typedef struct{
    BYTE name[64];
    BYTE Attr;
    DWORD FstClus;
    DWORD FileSize;
}FatElement;

```

```
typedef struct{
    FatElement parent;
    BYTE name[64];
    DWORD FstClus;
    FatElement content[20];
    int elementCount;
}FATDir;

typedef BYTE SECTOR[512];
typedef BYTE fatENTRY[32];

void initializeFat32(void);
BYTE getDataSector(SECTOR* sector,DWORD sectorNum);
BYTE getSector(SECTOR* sector,DWORD sectorNum);
void getRootDirectory(FATDir* dir);
DWORD getFATNextSector(DWORD sector);
void getDirectoryContent(DWORD address,FATDir* dir);
bool extensionCheck(FatElement element);

#endif /* _FATFS */
```

```
#include "FAT32.h"
#include "mmc_sd.h"
#include <stdbool.h>

WORD sectorSize;
BPB bpb;
DWORD bootDriveOffset;
DWORD dataOffset;

void initializeFat32(void){ //get the BPB region
    SECTOR temp;
    getSector(&temp,0); // load first sector in the temp variable.
    first sector is boot sector not BPB
    bootDriveOffset=LD_DWORD(&temp[454]); //the DWORD in the 454
    position of the boot sector contains the sector of the BPB
    getSector(&temp,bootDriveOffset); //get the bpb sector

    bpb.BytsPerSect=LD_WORD(&temp[BPB_BytsPerSec]); //save the usefull
    information
    bpb.SecPerClus=temp[BPB_SecPerClus];
    bpb.RsvdSecCnt=LD_WORD(&temp[BPB_RsvdSecCnt])+ bootDriveOffset;
    bpb.NumFATs=temp[BPB_NumFATs];
    bpb.FATSz32=LD_DWORD(&temp[BPB_FATSz32]);
    bpb.RootClus=LD_DWORD(&temp[BPB_RootClus]);

    dataOffset=bpb.RsvdSecCnt+bpb.FATSz32*bpb.NumFATs;
}

u8 getSector(SECTOR* sect,DWORD sector){ //get sector from SD Card
    return SD_ReadDisk(sect[0],sector,1);
}

u8 getDataSector(SECTOR* sect,DWORD sector){
    DWORD sectorNum;
    if(sector>1) sector=sector-2; // data sectors start at fat sector
    2, so sd sector 0 contains fat sector 2
    sectorNum=dataOffset+(sector)*bpb.SecPerClus;
    return getSector(sect,sectorNum);
}

DWORD getFATNextSector(DWORD sector){ //recover from the FAT table the
    next sector if there is any
    DWORD fatSector;
    UCHAR offset;
    SECTOR FATpart;
    if(sector<2) sector=2;
    fatSector=bpb.RsvdSecCnt+4*sector*bpb.SecPerClus/bpb.BytsPerSect;
    offset=4*sector%bpb.BytsPerSect;
    if(getSector(&FATpart,fatSector)!=0){
        return 0x0FFFFFFFA;
    }
    return LD_DWORD(&FATpart[offset]);
}

void getEntry(SECTOR sector,BYTE* entry,BYTE n){ // get the n 32bit
    entry from a directory sector
    int i,offset=n*32;
    for(i=0;i<32;i++){
        *entry=sector[i+offset];
    }
}
```

```

        entry++;
    }
}

void getDirectoryContent(DWORD address, FATDir* dir) {
    SECTOR directory;
    FatElement element;
    fatENTRY entry;
    int i, j, k;
    int rem;
    FatElement el;
    for(i=0; i<20; i++) { //initialize all elements' name
        dir->content[i]=el;
    }
    dir->elementCount=0;

    getDataSector(&directory, address);

    for(i=0; i<16; i++) {
        for(j=0; j<64; j++) { //we fill the name with eoc value
            element.name[j]=0xff;
        }

        getEntry(directory, &entry[0], i);
        if(entry[0]==0xE5) continue;
        if(entry[0]==0) return;

        if(entry[11]==0x0f && entry[0]&0x40) { //if the entry is the first
of a long name sequence
            rem=entry[0]&0x3f; //we get the number of long name
entries

            for(j=rem-1; j>=0; j--) { //for each long name entry we
get the name characters
                for(k=0; k<5; k++) {
                    element.name[j*13+k]=entry[1+2*k];
                }
                for(k=5; k<11; k++) {
                    element.name[j*13+k]=entry[4+2*k];
                }
                element.name[j*13+11]=entry[28];
                element.name[j*13+12]=entry[30];
                i++;
                getEntry(directory, &entry[0], i); //and get the next
entry
            }
        }
        if(entry[11]>0x0f) {
            if(element.name[0]==0xff) { //if the entry didnt had a long
name we put the short name
                for(j=0; j<11; j++) {
                    element.name[j]=entry[j];
                }
            }
            element.Attr=entry[11];

            element.FstClus=(LD_WORD(&entry[20])<<16 | LD_WORD(&entry[26]));
            element.FileSize=LD_DWORD(&entry[28]);

```

```
        dir->content[dir->elementCount++]=element;
    }
    if(i==15){ //if the directory continues in other sectors reload
the loop
        DWORD next=getFATNextSector(address);
        if(next<0xffff7){
            getDataSector(&directory,next);
            i=-1;
        }
    }
}

void getRootDirectory(FATDir* dir){
    getDirectoryContent(bpb.RootClus, dir);
}
```

## Annex 6.3 SD Card

```
#include "sys.h"
#include "mmc_sd.h"
#include "spi.h"
#include "usart.h"

#define SD_TYPE_ERR      0X00  // define constant values for sd card
com.
#define SD_TYPE_MMC      0X01
#define SD_TYPE_V1       0X02
#define SD_TYPE_V2       0X04
#define SD_TYPE_V2HC     0X06

#define CMD0      0
#define CMD1      1
#define CMD8      8
#define CMD9      9
#define CMD10     10
#define CMD12     12
#define CMD16     16
#define CMD17     17
#define CMD18     18
#define CMD23     23
#define CMD24     24
#define CMD25     25
#define CMD41     41
#define CMD55     55
#define CMD58     58
#define CMD59     59

#define MSD_DATA_OK                0x05
#define MSD_DATA_CRC_ERROR        0x0B
#define MSD_DATA_WRITE_ERROR      0x0D
#define MSD_DATA_OTHER_ERROR      0xFF

#define MSD_RESPONSE_NO_ERROR     0x00
#define MSD_IN_IDLE_STATE         0x01
#define MSD_ERASE_RESET           0x02
#define MSD_ILLEGAL_COMMAND       0x04
#define MSD_COM_CRC_ERROR         0x08
#define MSD_ERASE_SEQUENCE_ERROR  0x10
#define MSD_ADDRESS_ERROR         0x20
#define MSD_PARAMETER_ERROR       0x40
#define MSD_RESPONSE_FAILURE      0xFF

#define SD_CS    PAout(4)

u8  SD_Type=0;

u8 SD_SPI_ReadWriteByte(u8 data) //sent the data in the argument and
returns the received data
{
    return SPIx_ReadWriteByte(data);
}

void SD_SPI_SpeedLow(void) //set the baud rate low enough for sd card
init
{
```



```
    SPIx_SetSpeed(SPI_BaudRatePrescaler_256);
}

void SD_SPI_SpeedHigh(void) //set a higher the baud rate to speed up
communication
{
    SPIx_SetSpeed(SPI_BaudRatePrescaler_16);
}

void SD_SPI_Init(void) //initialize the spi peripheral bus
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA, ENABLE );

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP ;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    GPIO_SetBits(GPIOA,GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4);

    SPIx_Init();
    SD_CS=1;
}

void SD_DisSelect(void) //disables the spi slave
{
    SD_CS=1;
    SD_SPI_ReadWriteByte(0xff);
}

u8 SD_Select(void) //enables the spi slave
{
    SD_CS=0;
    if(SD_WaitReady()==0) return 0;
    SD_DisSelect();
    return 1;
}

u8 SD_WaitReady(void)
{
    u32 t=0;
    do
    {
        if(SD_SPI_ReadWriteByte(0xFF)==0xFF) return 0; // sent dummy
bytes until sd responds or timeout
        t++;
    }while(t<0xFFFF);
    return 1;
}

u8 SD_GetResponse(u8 Response)
{
    u16 Count=0xFFF; //sends a dummy byte and reads
the response
    while ((SD_SPI_ReadWriteByte(0xFF)!=Response)&&Count) Count--;
    if (Count==0) return MSD_RESPONSE_FAILURE; //if response is not
what expected or no response return with error
    else return MSD_RESPONSE_NO_ERROR;
}
```

```
}

u8 SD_RecvData(u8*buf,u16 len)
{
    if(SD_GetResponse(0xFF)) return 1; //if the response is not the data
    token return
    while(len--)
    {
        *buf=SPIx_ReadWriteByte(0xFF); //stores in the buffer all the
        received data
        buf++;
    }
    SD_SPI_ReadWriteByte(0xFF); //wait for two dummy bytes linked to
    the read command to avoid interference in the next command
    SD_SPI_ReadWriteByte(0xFF);
    return 0;
}

u8 SD_SendCmd(u8 cmd, u32 arg, u8 crc)
{
    u8 r1;
    u8 Retry=0;
    SD_DisSelect();
    if(SD_Select()) return 0xFF;

    SD_SPI_ReadWriteByte(cmd | 0x40); // send the command code with the
    mask
    SD_SPI_ReadWriteByte(arg >> 24); //sen the argument bytes
    SD_SPI_ReadWriteByte(arg >> 16);
    SD_SPI_ReadWriteByte(arg >> 8);
    SD_SPI_ReadWriteByte(arg);
    SD_SPI_ReadWriteByte(crc); //send the crc
    if(cmd==CMD12) SD_SPI_ReadWriteByte(0xff); //Skip a stuff byte when
    stop reading
    Retry=0x1F;
    do
    {
        r1=SD_SPI_ReadWriteByte(0xFF); //read bytes until get a valid
        response (start with 0)
    }while((r1&0x80) && Retry--);

    return r1; //return response
}

u8 SD_Initialize(void)
{
    u8 r1;
    u16 retry;
    u8 buf[4];
    u16 i;

    SD_SPI_Init();
    SD_SPI_SpeedLow();

    for(i=0;i<10;i++) SD_SPI_ReadWriteByte(0xFF); //make the sd card
    enter SPI mode
    retry=5;
    do//initialization proces explained in the corresponding section
```

```

{
    r1=SD_SendCmd(CMD0,0,0x95);
}while((r1!=0X01) && retry--);
SD_Type=0;
if(r1==0X01)
{
    if(SD_SendCmd(CMD8,0x1AA,0x87)==1) //SD V2.0
    {
        for(i=0;i<4;i++)buf[i]=SD_SPI_ReadWriteByte(0XFF); //Get
trailing return value of R7 resp
        if(buf[2]==0X01&&buf[3]==0XAA)
        {
            retry=0XFFE;
            do
            {
                r1=SD_SendCmd(CMD55,0,0X01);
                r1=SD_SendCmd(CMD41,0x40000000,0X01);
            }while(r1&&--retry);
            if(retry&&SD_SendCmd(CMD58,0,0X01)==0)
            {
                for(i=0;i<4;i++)buf[i]=SD_SPI_ReadWriteByte(0XFF);
                if(buf[0]&0x40)SD_Type=SD_TYPE_V2HC;
                else SD_Type=SD_TYPE_V2;
            }
            else{
                retry=0XFFE;
                do
                {
                    r1=SD_SendCmd(CMD1,0,0X01);
                }while(r1&&--retry);
            }
        }
    }else//SD V1.x/ MMC V3
    {
        SD_SendCmd(CMD55,0,0X01);
        r1=SD_SendCmd(CMD41,0,0X01);
        if(r1<=1)
        {
            SD_Type=SD_TYPE_V1;
            retry=0XFFFE;
            do
            {
                SD_SendCmd(CMD55,0,0X01);
                r1=SD_SendCmd(CMD41,0,0X01);
            }while(r1&&retry--);
        }else
        {
            SD_Type=SD_TYPE_MMC;//MMC V3
            retry=0XFFFE;
            do
            {
                r1=SD_SendCmd(CMD1,0,0X01);
            }while(r1&&retry--);
        }
    }

if(retry==0||SD_SendCmd(CMD16,512,0X01)!=0)SD_Type=SD_TYPE_ERR;
}

SD_DisSelect();
SD_SPI_SpeedHigh();
if(SD_Type)return 0;

```

```
    else if(r1)return r1;
    return 0xaa; //return in case init ok
}

u8 SD_ReadDisk(u8*buf,u32 sector,u8 cnt) //stores the content of a card
sector or sectors in the buffer
{
    u8 r1;
    if(SD_Type!=SD_TYPE_V2HC)sector <<= 9; //vhc2 addresses by sector,
if not that card needs to be multiplied by 512 to simulate address by
sector
    if(cnt==1)
    {
        r1=SD_SendCmd(CMD17,sector,0x01); // command for reading one
sector
        if(r1==0)
        {
            r1=SD_RecvData(buf,512);
        }
    }else
    {
        r1=SD_SendCmd(CMD18,sector,0x01); // comand for reading
multiple sectors
        do
        {
            r1=SD_RecvData(buf,512);
            buf+=512;
        }while(--cnt && r1==0);
        SD_SendCmd(CMD12,0,0x01); //command to stop reading
    }
    SD_DisSelect();
    return r1;
}
```

## Annex 6.4 Fat directory- lcd interface

```
#include "FAT32.h"
#include "GUI.h"
#include "lcd.h"
#include <stdbool.h>
//constant for positioning in the lcd
u16 padding=20;
u16 currentFolder=50;
u16 separator1y=75, separator2y=280;
u16 separatorHeight=5;
u16 file=95,fOffset=30;
u16 error=295;

//variables
u8 maxFolders=6;

FatElement dirContent[20];
u16 highlitghPos;
u16 nDir;

void writeCurrentFolder(u8* str){// print the current directory name
    u8 s[100]="Current folder: ";
    int i=0;
    while(*str!=0&&*str!=0xff){
        s[16+i]=*str; //concatenates the strings
        i++;
        str++;
    }

    Show_Str(padding,currentFolder,WHITE,BLACK,s,16,0);
}

void LoadMenuInit(void){
    u8 s[]="G-ENGINE";//prints the title and the separators

    LCD_Fill(0,0,lcddev.width,lcddev.height,BLACK);

    Show_Str(padding,padding,BLUE,BLACK,s,16,0);
    writeCurrentFolder((u8*) "NAN");

    LCD_DrawFillRectangle(0,separator1y,lcddev.width,separator1y+separator
Height);

    LCD_DrawFillRectangle(0,separator2y,lcddev.width,separator2y+separator
Height);

}

void writeError(u8* str){ // prints an error string in red

    LCD_Fill(0,separator2y+separatorHeight,lcddev.width,lcddev.height,BLAC
K);
    if(str!=0) Show_Str(padding,error,RED,BLACK,str,16,0);
}

void writeDirSimbol(u8 pos,bool highlight){// prints the "/" before the
directory name
```

```
        u16 color;
        u8 content=0x5c;
        if(higlight) color=BLUE;
        else color=WHITE;
        Show_Str(padding-
8,file+pos*fOffset,color,BLACK,&content,16,0);

    }

void writeSingleContent(u8 pos,u8* content,bool higlight){ // prints a
element of the directory, in colour blue if its highlighted
    u16 color;
    if(higlight) color=BLUE;
    else color=WHITE;
    Show_Str(padding,file+pos*fOffset,color,BLACK,content,16,0);

}

void writeContent(FatElement content[],u16 highlighted){
    int i;
    int offset=highlighted/6;
    highlighted=highlighted%6;//if there are more elements that can
fit, adjust with these variables

    LCD_Fill(0,separatorly+separatorHeight,lcddev.width,separator2y,BLACK)
; // puts in black all pixel in the directory section
    for(i=0;i<maxFolders;i++){ // show the 6 elements that fit
        if(content[i+offset*6].name==0) continue;

        if(content[i+offset*6].Attr==0x10||content[i+offset*6].Attr==0x11){
            writeDirSimbol(i,i==highlighted);
        }
        writeSingleContent(i,content[i+offset*6].name,i==highlighted);

    }

}

void loadContent(FATDir* dir){ // save from the FATDir the elements
needed, directories and files that are visible
    int i,j;
    FatElement el;
    for(i=0;i<64;i++){//initialize variables
        el.name[i]=0;
    }
    for(i=0;i<20;i++){
        dirContent[i]=el;
    }

    for(j=i=0;i<20;i++){
        FatElement element=dir->content[i];

        if(element.Attr==0x10||element.Attr==0x11||element.Attr==0x20||element
.Attr==0x21){// is element is directory or file either readonly or not
            dirContent[j]=element;//save to internal variable
            j++;
        }

    }

}
```

```
        nDir=j;
        writeContent(dirContent,highlitghPos=0); // show the new loaded
        directory on the lcd
    }

    void goUp(void){//highlight the above item

        if(highlitghPos>0){
            highlitghPos--;
            writeContent(dirContent,highlitghPos);
        }
    }

    void goDown(void){//highlight the below item

        if(highlitghPos<nDir-1){
            highlitghPos++;
            writeContent(dirContent,highlitghPos);
        }
    }

    FatElement open(void){//return the highlight item

        return dirContent[highlitghPos];
    }
}
```

## Annex 6.5 controller input gathering

```
#include "Controller.h"

#include "Timer2.h"

#include <stdbool.h>

GPIO_TypeDef* Port;

Buttons controlador;

uint32_t minDebounce= 8u;

uint32_t lastDebounceTime=0;

uint16_t buttonState=0;

uint16_t lastButtonState=0;

void ControllerInit(GPIO_TypeDef* ControllerPort,uint32_t
ControllerPortCLK){

    //GPIO structure used to initialize port

    GPIO_InitTypeDef GPIO_InitStructure;

    //Enable clock on APB2 peripheral bus where button and LEDs are
connected

    RCC_APB2PeriphClockCmd(ControllerPortCLK,  ENABLE);

    //select pins to initialize button

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;

    //highest speed available

    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

    //select input pulldown

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;

    GPIO_Init(ControllerPort, &GPIO_InitStructure);

    Port=ControllerPort;// save the gpio port for later use

}
```



```
uint16_t getButtonState(){ //gets the input and debounce it

    uint16_t newState,result;

    uint16_t lastState=lastButtonState;

    bool validState=false;

    while (!validState){

        newState=GPIO_ReadInputData(Port); // save the current state of
the buttons

        if(newState!=lastState){ // if current state is different than
the last state, update state and reference time

            lastState=newState;

            lastDebounceTime=millis();

        } else if(millis()-lastDebounceTime>minDebounce){ //if the
state has not changed in a while accept the current state as valid
State

            validState=true;

        }

    }

    result=newState&0xff; //save the 8 low bits

    if(lastButtonState!=newState){ // if last valid state is different
than the new one activate statechange flag

        result|=1<<8u;

        lastButtonState=newState;

    }

    if(result&0x100){ //if statechange flag update all states

        controlador.stateChange=true;

        controlador.actionB=0x001&result;

        controlador.actionA=0x001&result>>1;

        controlador.start=0x001&result>>2;

        controlador.select=0x001&result>>3;

        controlador.right=0x001&result>>4;
```

```
        controlador.down=0x001&result>>5;

        controlador.left=0x001&result>>6;

        controlador.up=0x001&result>>7;

    }else{

        controlador.stateChange=false;

    }

    return result;
}

//return of all button states
bool getLeft(void) {

    return controlador.left;

}

bool getRight(void) {

    return controlador.right;

}

bool getUp(void) {

    return controlador.up;

}

bool getDown(void) {

    return controlador.down;

}

bool getStateChange(void) {

    return controlador.stateChange;

}

bool getActionA(void) {

    return controlador.actionA;

}

bool getActionB(void) {

    return controlador.actionB;
```

```
}  
  
bool getSelect(void) {  
    return controlador.select;  
}  
  
bool getStart(void) {  
    return controlador.start;  
}
```

## Annex 6.6 Game code

```
#include "delay.h"

#include "sys.h"

#include "lcd.h"

#include "gui.h"

#include "sprites.h"

#include "controller.h"

#include "Timer2.h"

#include <stdbool.h>

/*****

    Sprite Struct
*****/

typedef struct{

    const unsigned char* bitmap;

    int x,y;//current position

    int xnew,ynew; //position on the next lcd update

    int width,height; //dimensions

    int scale;

    int xspd,yspd; //speed x and y

    bool isVisible; // whether or not must be printed on screen or
taken care in collisions

    bool hasSpeed; // for code friendliness same as xspd!=0&&yspd!=0

    bool needUpdate; //sprite changed somehow since last lcd update

    bool outlimits; //sprite will be out of LCD border next update

    uint32_t timeToDie; // for aliens only, time at which it will
become not visible

}Sprite;

/*****
```

```
Game variables

*****/

bool modeA=true;//alternate between the two alien sprites

bool directionRight=true;//alternate between alien movement to the
right or to the left

int scale=2;// scale for every sprite

bool left,right,actionA,start,select,stateChange;//variables where
pushed button is saved

int score=0;

int lives=3;

const int maxAliens=24;

const int maxShields=4;

const int maxAliensShots=3;

const int maxSprites=(1+maxAliens+maxAliensShots+maxShields+1); //
number of all sprites including player and playershot sprites

int aliensRemaining;

bool gameOver;

uint32_t lastSpeedUpdate=0;

uint32_t lastAlienUpdate=0;

/*****

    Sprites

*****/

int spriteCount=0;

Sprite* allSprites[maxSprites];

int shieldLives[5];//how many shots can stand the shields

Sprite player;

Sprite aliens[maxAliens];

Sprite shields[maxShields];

Sprite shot;
```

```
Sprite enemyShot[maxAliensShots];

/*****

    Methods

*****/

Sprite SpriteInit(const unsigned char* bmp,int x,int y,int scale);void
createShieldRow(void);

void createAliensRow(void);

void printScreen(void);

void printHeader(void);

void readAction(void);

void updateGame(void);

void startGame(void);

void pause(void);

void initVariables(void){ //initialize all game variables

    int i,j;

    srand(millis()); //give a seed to the RNG

    allSprites[i]=&player;

    i++;

    aliensRemaining=0; //initialize variable

    for(j=0;j<maxShields;j++){ //assign pointers to the all sprite
array
        allSprites[i]=&shields[j];

        i++;

    }

    for(j=0;j<maxAliens;j++){

        allSprites[i]=&aliens[j];

        i++;

    }

    allSprites[i]=&shot;
```

```
        i++;

        for(j=0;j<maxAliensShots;j++){

            allSprites[i]=&enemyShot[j];

            i++;

        }

    }

}

int Game(void)//equivalent to the main of the game code
{

    SystemInit();

    ControllerInit(GPIOE,RCC_APB2Periph_GPIOE);

    Timer2_init();

    delay_init(72);

    LCD_Init();

    LCD_Fill(0,0,lcddev.width,lcddev.height,BLACK);//fill all screen
    black

    initVariables();

    player=SpriteInit(gImage_player,0,lcddev.height-
    *(gImage_player+4)*scale,scale); //initialize player sprite at the
    bottom left corner

    createAliensRow(); // initialize alien sprites

    createShieldRow();// initialize shield sprites

    printHeader(); // print HUD structure

    printScreen(); //print all sprites

    while(1)//infinite loop of the game
    {

        readAction();//collect input

        updateGame(); //update game content

        printScreen(); // print all updated sprites
```

```
    }

}

void change_Sprite(Sprite* sp, const unsigned char* image){ //change a
sprite display image and redimension

    int scale=sp->scale;

    sp->bitmap=image+8;

    sp->width=*(image+2)*scale;

    sp->heigh=*(image+4)*scale;

    sp->needUpdate=true;

}

void startGame() {

    int i;

    for(i=0;i<maxShields;i++){

        shieldLifes[i]=0;// restore all shields lifes

    }

    score=0; //score reset

    change_Sprite(&player,gImage_player); // change player sprite to
the alive sprite

    vidas=3; //restore lifes

    LCD_Fill_Off(0,0,lcddev.width,lcddev.height,N); //reset screen

    createAliensRow();

    createShieldRow();

    printHeader();

    printScreen();

    gameOver=false;

}

Sprite SpriteInit(const unsigned char* bmp,int x,int y,int scale){//
create sprite and return it
```



```
Sprite sp; //fills all sprite fields

sp.bitmap=bmp+8;

sp.width=(bmp+2)*scale;

sp.heigh=(bmp+4)*scale;

sp.scale=scale;

sp.x=x;

sp.xnew=x;

sp.y=y;

sp.ynew=y;

sp.xspd=0;

sp.yspd=0;

sp.hasSpeed=false;

sp.needUpdate=true;

sp.isVisible=true;

sp.timeToDie=0xffffffff;

return sp;
}

void draw_sprite(Sprite sp){ //print the sprite in the lcd mehthod
from

    draw_sprite_uChar_scaled( sp.x,
sp.y,sp.width/sp.scale,sp.heigh/sp.scale,sp.bitmap,sp.scale);

}

void setSpeed(Sprite *player,int speedx,int speedy){//change sprite
speed

    if(speedx==0&&speedy==0) player->hasSpeed=false;

    else player->hasSpeed=true;

    player->xspd=speedx;

    player->yspd=speedy;
```

```
}

void refillBackground(Sprite sp){ //print with background colour the
area no longer occupied by a moving sprite

    int incX=sp.xnew-sp.x;

    int incY=sp.ynew-sp.y;

    if(incX==0&&incY==0) return;

    if(incX>=sp.width||-incX>=sp.width||incY>=sp.heigh||-
incY>=sp.heigh){//if there is no common pixels between the two
locations fill all the previous area

        LCD_Fill_Off(sp.x,sp.y,sp.width,sp.heigh,N);

        return;

    }

    if(incX>0){//otherwise calculate the area that is in common and
fill the rest

        LCD_Fill_Off(sp.x,sp.y,incX,sp.heigh,N);

        if(incY>0){

            LCD_Fill_Off(sp.xnew,sp.y,sp.width-incX,incY,N);

        }else if(incY<0){

            LCD_Fill_Off(sp.x,sp.ynew+sp.heigh,sp.width-incX,-
incY,N);

        }

    }else if(incX<0){

        LCD_Fill_Off(sp.xnew+sp.width,sp.y,-incX,sp.heigh,N);

        if(incY>0){

            LCD_Fill_Off(sp.x,sp.y,sp.width-incX,incY,N);

        } else if(incY<0){

            LCD_Fill_Off(sp.xnew,sp.ynew+sp.heigh,sp.width-incX,-
incY,N);

        }

    }else{
```

```
        if(incY>0){

            LCD_Fill_Off(sp.x,sp.y,sp.width-incX,incY,N);

        } else {

            LCD_Fill_Off(sp.x,sp.ynew+sp.heigh,sp.width,-incY,N);

        }

    }

}

void removeSprite(Sprite* sp){ // removes a sprite from the game, set
the sprite visibility to to false and initilize its fields

    LCD_Fill_Off(sp->x,sp->y,sp->width,sp->heigh,N);

    sp->isVisible=false;

    sp->timeToDie=0xffffffff;

    sp->needUpdate=false;

    setSpeed(sp,0,0);

}

void updateHeader(void){ //update the HUD content

    LCD_ShowNum(70,20,vidas,1,16);

    LCD_ShowNum(400,20,score,6,16);

}

void printScreen(){

    int i;

    Sprite* sp;

    updateHeader();

    for(i=0;i<maxSprites;i++){//goes through all the sprites that need
update
```

```
        sp=allSprites[i];

        if(sp->needUpdate){

            if(!sp->isVisible){//if sprite is not visible and needs an
update removes it

                removeSprite(sp);

            }else{//otherwise fills the previous location with
background colour and update the positions

                refillBackground(*sp);

                sp->x=sp->xnew;

                sp->y=sp->ynew;

                draw_sprite(*sp);

                sp->needUpdate=false;

            }

        }

    }

}
```

```
bool collision(Sprite sp,Sprite sp2){ // checks collision between 2
sprites

    int
    p1x1=sp.xnew,p1x2=sp.xnew+sp.width,ply1=sp.ynew,ply2=sp.ynew+sp.heigh;

    int
    p2x1=sp2.x,p2x2=sp2.x+sp2.width,p2y1=sp2.y,p2y2=sp2.y+sp2.heigh;//
define variables representing the four corners of the sprite

    bool x=false,y=false; //variables that store if there is
superposition in these axis.

    if(!sp.isVisible)return false;

    if(!sp2.isVisible)return false; //invisible sprites do not collide

    if(p1x1>p2x2){//calculates superposition of axis

        x=false;
```

```
    }else    if(p1x2<p2x1){  
        x=false;  
    }else x=true;  
  
    if(p1y1>p2y2){  
        y=false;  
    }else    if(p1y2<p2y1){  
        y=false;  
    }else y=true;  
  
    return x&& y; // collision exist when there is superposition in  
both axis  
}
```

```
bool changePos(Sprite *p,int x,int y){// change the next update  
position if sprite doesn't surpass the edges of screen  
  
    bool validMove=true;  
  
    Sprite p2=*p;  
  
    if(x<0){  
        x=0;  
        validMove=false;  
    }  
  
    if(p2.width+x>lcddev.width){  
        x=lcddev.width-p2.width;  
        validMove=false;  
    }  
  
    if(y<0){
```

```
        y=0;

        validMove=false;

    }

    if(y+p2.heigh>lcddev.height){

        y=lcddev.height-p2.heigh;

        validMove=false;

    }


    p->xnew=x;

    p->ynew=y;

    p->needUpdate=true;

    return validMove;

}


bool setPos(Sprite *p,int x,int y){

    bool validMove=changePos(p,x,y);

    refillBackground(*p);

    p->x=x;

    p->y=y;

    return validMove;

}


void shoot(){// creates a shot from the player with vertical speed

    if(!shot.isVisible){

        shot=SpriteInitIndex(gImage_shot,0,0,scale,31);

        setPos(&shot,player.x+player.width/2-shot.width/2,player.y-
shot.heigh+1);

    }

}
```

```
        setSpeed(&shot,0,-10);

    }

}

void readAction(void) {

    getButtonState();

    stateChange=getStateChange();

    left=getLeft();

    right=getRight();

    actionA=getActionA();

    start=getStart();

    select=getSelect();//get the controller input and set
horizontal speed, create shot, or pause the game

    if(left){

        setSpeed(&player,-6,0);

    }else if(right){

        setSpeed(&player,6,0);

    }else setSpeed(&player,0,0);

    if(actionA){

        shoot();

    }

    if(select&&stateChange){

        pause();

    }

}

void createAliensRow(void) {
```

```
    int i;

    int xinc=lcddev.width/10;

    int yinc=lcddev.height/10;

    const unsigned char* im=gImage_alien1a;

    int xin=xinc-*(im+2)*3/2;

    //create the alien rows with the different type of ships

    for(i=0;i<8;i++){

        aliens[i]=SpriteInit(im,xin+i*xinc,2*yinc,scale);

        aliensRemaining++;

    }


    im=gImage_alien2a;

    xin=xinc-*(im+2)*3/2;

    for(i=0;i<8;i++){

        aliens[i+8]=SpriteInit(im,xin+i*xinc,3*yinc,scale);

        aliensRemaining++;

    }

    im=gImage_alien3a;

    xin=xinc-*(im+2)*3/2;

    for(i=0;i<8;i++){

        aliens[i+16]=SpriteInit(im,xin+i*xinc,4*yinc,scale);

        aliensRemaining++;

    }

}


void createShieldRow(void){//creates the shield row

    int i;

    int xinc=lcddev.width/5;

    const unsigned char* im=gImage_shield100;

    int yinc=player.y-*(im+4)*2*scale;
```



```
int xin=xinc-*(im+2)*3/2;

for(i=0;i<4;i++){

    shields[i]=SpriteInit(im,xin+i*xinc,yinc,scale);

}

}

bool outLimits(Sprite sp){// given a sprite returns whether or not its
new position will be out of bounds

int
plx1=sp.xnew,plx2=sp.xnew+sp.width,ply1=sp.ynew,ply2=sp.ynew+sp.heigh;

if(plx1<0)return true;

if(plx2>lcddev.width)return true;

if(ply1<55) return true;

if(ply2>lcddev.height)return true;

return false;

}

void showMessage(u8* s1, u8* s2){ //Creates a message screen with two
lines of text, used for game over, pause and next round

LCD_Fill_Off(120,80,240,160,WHITE);

LCD_Fill_Off(130,90,220,140,BLACK);

Gui_StrCenter(0,140-16,WHITE,BLACK,s1,16,0);

Gui_StrCenter(0,140+16,WHITE,BLACK,s2,16,0);

}

void GameOver(void){

u8 s1[]="GAME";

u8 s2[]="OVER";

showMessage(s1,s2);

gameOver=true;

while(gameOver){ // game over wait until a press of start button
to start a new game
```

```
        getButtonState();

        if(getStart()){

            startGame();

        }

    }

}

void pause(){

    bool paused=true;

    u8 s1[]=" ";

    u8 s2[]="PAUSED";

    showMessage(s1,s2);

    while(paused){ //game paused until pause button is pressed again

        getButtonState();

        select=getSelect();

        stateChange=getStateChange();

        if(select&&stateChange) paused=false;

    }

    LCD_Fill_Off(120,80,240,160,BLACK);//eliminates the message box

    printScreen();//print again all sprites

}

void updateAlienPos(void){

    int i;

    int movement[2]; //mov[0]=x mov[1]=y

    const unsigned char* sprites[3]; // will contain the images
for the 3 types of alien ships

    if(modeA){ //use model A sprites

        sprites[0]=gImage_alien1a;

        sprites[1]=gImage_alien2a;
```

```
        sprites[2]=gImage_alien3a;

    }else{// use model B sprites

        sprites[0]=gImage_alien1b;

        sprites[1]=gImage_alien2b;

        sprites[2]=gImage_alien3b;

    }

    if(directionRight){//set the movement to the right or left
by 10 pixels

        movement[0]=+10;

        movement[1]=0;

    }

    else{

        movement[0]=-10;

        movement[1]=0;

    }

    for(i=0;i<sizeof(alien)/sizeof(alien[0]);i++){//for all
alien sprites

    if(alien[i].isVisible&&alien[i].timeToDie==0xffffffff){//only for
alive aliens

        change_Sprite(&alien[i],sprites[i/8]);// assign new
sprite image according to row

    if(!changePos(&alien[i],alien[i].x+movement[0],alien[i].y+movement[
1])){//if an alien new position became out of bounds

        i=-1;//start over for loop. the -1 value is to
deal with i++

        directionRight=!directionRight; //change
directionRight

        movement[0]=0;// move down instead of laterally

        movement[1]=8;

    }else{//if alien is not out of bounds check if alien
arrived at shield heigh, if it did means game over
```

```
        if((aliens[i].y+aliens[i].heigh)>shields[0].y){

            change_Sprite(&player,gImage_playerDead); //
change image to show the played dead and go to game over

            draw_sprite(player);

            GameOver();

        }

    }

}

modeA=!modeA; //after all aliens moved change the ship model

}

void updateSpritesPos(void){

    int i;

    for(i=0;i<maxSprites;i++){

        Sprite* sp=allSprites[i];

        if(sp->hasSpeed){ // for all the sprites that have speed

            int x=sp->x+sp->xspd;

            int y=sp->y+sp->yspd;

            if(sp->xspd!=0){ //change sprites' next position

                sp->xnew=x;

                sp->needUpdate=true;

            }else{

                sp->ynew=y;

                sp->needUpdate=true;

            }

        }

    }

}
```

```
        }

    }

    if(sp->timeToDie<millis()){//if it is time to die for an
alien remove it

        removeSprite(sp);

    }

    if(outLimits(*sp)){// if the new pos is out of bounds
return to previous state

        sp->xnew=sp->x;

        sp->ynew=sp->y;

        sp->outlimits=true;

        sp->needUpdate=false;

    }else if(sp->outlimits) sp->outlimits=false;

}

lastSpeedUpdate=millis(); //update last update time
}

void checkPlayerShotCollision(void){

    if(shot.isVisible){

        int i;

        for(i=0;i<sizeof(alien)/sizeof(alien[0]);i++){ //for all
alien

            if(collision(shot,alien[i])){//if there is collision with
the player shot

                removeSprite(&shot); //eliminate shot sprite

                change_Sprite(&alien[i],gImage_alienDead); //chane
alien image to dead alien

                alien[i].timeToDie=millis()+150; // set alien time to
die 150 milliseconds in the future

                score+=10*(3-i/8); //increase score according to alien
row

                alienRemaining--; // decrease alien count

                if(alienRemaining==0){ //if all aliens eliminated
```

```
        draw_sprite(alien[i]); // print the new sprite

        delay(150);

        removeSprite(&alien[i]); //wait for time to die

        showMessage((u8*)"NEXT ROUND", (u8*)"PRESS
START"); // show next round message

        while(!getStart()){

            getButtonState(); // wait until start pressed
for next round

        }

        LCD_Fill_Off(120,80,240,160,BLACK); //remove
message box and reset game parameters for next round

        score+=150;

        directionRight=true;

        createAliensRow();

    }

}

    for(i=0;i<sizeof(shields)/sizeof(shields[0]);i++){ //if the
shot hits a shield subtract a life and update its sprite to show
damage

        if(collision(shot,shields[i])){

            removeSprite(&shot);

            switch(++shieldLives[i]){

                case 1:
change_Sprite(&shields[i],gImage_shield80);

                    break;

                case 2:
change_Sprite(&shields[i],gImage_shield60);

                    break;

                case 3:
change_Sprite(&shields[i],gImage_shield40);
```

```
                break;

                case 4:
change_Sprite(&shields[i],gImage_shield20);

                break;

                case 5: shields[i].needUpdate=true;

                        shields[i].isVisible=false;

                break;

        }

    }

}

    if(shot.y<=65){// if shot arrives at the top without colliding
eliminate it

        removeSprite(&shot);

    }

}

}

void createEnemyShot(Sprite p1,int index){ //create a shot in a alien
position with vertical speed towards ground

    enemyShot[index]=SpriteInit(gImage_shot,0,0,scale);

    setPos(&enemyShot[index],p1.x+p1.width/2-
enemyShot[index].width/2,p1.y+p1.heigh+1);

    setSpeed(&enemyShot[index],0,10);

}

void alienShot(void){

    int i,j;

    for(i=0;i<sizeof(enemyShot)/sizeof(enemyShot[0]);i++){// for
every alien shot

        if(!enemyShot[i].isVisible){// if the shot is not already
on screen
```

```
        if(rand()%5==0){// there is 1/20 chance that an alien
will shoot

                int index=rand()%8;//if an alien shot, makes a
random to determine from which column they'll shoot

                for(j=1;j<4;j++){//for all the aliens in that
column

                        Sprite sp=aliens[index+8*(3-j)]; //starting
from the bottom row, check if the alien is alive

if(sp.isVisible&&sp.timeToDie==0xffffffff){//if the alien is alive
shoot and end for loop

                                createEnemyShot(sp,i);

                                j=4;

                                }// if not alive the shot is wasted

                        }

                }

        }

}

void    checkAlienShotCollision(void) {

        int i,j;

        for(i=0;i<sizeof(enemyShot)/sizeof(enemyShot[0]);i++){// for
every alien shot that is visible

                if(enemyShot[i].isVisible) {

for(j=0;j<sizeof(shields)/sizeof(shields[0]);j++){// check collision
and update shield lifes

                        if(collision(enemyShot[i],shields[j])){

                                removeSprite(&enemyShot[i]);

                                switch(++shieldLifes[j]){

                                        case 1:

change_Sprite(&shields[j],gImage_shield80);

                                                break;
```



```
                                case 2:
change_Sprite(&shields[j],gImage_shield60);

                                break;

                                case 3:
change_Sprite(&shields[j],gImage_shield40);

                                break;

                                case 4:
change_Sprite(&shields[j],gImage_shield20);

                                break;

                                case 5: shields[j].needUpdate=true;

shields[j].isVisible=false;

                                break;

                                }

                                }

                                if(collision(enemyShot[i],player)){// if there
is collision with the player

                                int i;

                                vidas--;

                                change_Sprite(&player,gImage_playerDead);

                                draw_sprite(player); // subtract one life
and update the player sprite to dead player

                                for(i=0;i<maxAliensShots;i++){ //remove
all alien shots

                                removeSprite(&enemyShot[i]);

                                }

                                updateHeader(); //update hud information
to show life decrease

                                delay(500);

                                if(vidas==0){

                                GameOver(); // if no more lifes game
over

                                }else{ //other wise blink sprite image
and continue
```

```
        for(i=0;i<3;i++){

LCD_Fill_Off(player.x,player.y,player.width,player.heigh,N);

            delay(200);

            draw_sprite(player);

            delay(200);

        }

        change_Sprite(&player,gImage_player);

    }

}

    if(enemyShot[i].outlimits){ // if enemy shot is out
limits remove sprite

        enemyShot[i].outlimits=false;

        removeSprite(&enemyShot[i]);

    }

}

}

}

}

void updateGame(void) {

    if(millis()-lastSpeedUpdate>30){ //update movement every 30 ms

        updateSpritesPos();

        checkPlayerShotCollision();

        checkAlienShotCollision();

        if(millis()-lastAlienUpdate>500){ //update alien position and
alien shot chance every 500 ms

            updateAlienPos();

            alienShot();

        }

    }

}
```

```
        lastAlienUpdate=millis();  
    }  
  
}  
  
}  
  
void printHeader(void){ // print HUD  
    u8 s[]="LIFES: ";  
    u8 s2[]="SCORE: ";  
    Show_Str(10,20,W,N,s,16,0);  
    LCD_ShowNum(70,20,vidas,1,16);  
    Show_Str(335,20,W,N,s2,16,0);  
    LCD_ShowNum(400,20,score,6,16);  
    LCD_Fill_Off(0,50,lcddev.width,4,W);  
}
```

## Annex 6.7 Timer 2 configuration

```
#include "Timer2.h"

uint16_t MSB=0; // variable to extend counter capacity

void NVIC_Configuration () { // interrupt configuration
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

    NVIC_Init(&NVIC_InitStructure);
}

void Timer2_init(void) {
    //Enabling the clock
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

    /* Set timer prescaler */
    TIM2->PSC = 36000u; //resolution of 500 microseconds and therefore
    update event every ~32 second

    /* Set reload register */
    TIM2->ARR = 0xFFFF;

    /* Enable the timer */
    TIM2->CR1 |= TIM_CR1_CEN;

    TIM2->CNT = 0u;

    TIM2->DIER|=1; //activate update interrupt
```

```
    NVIC_Configuration ();

}

void overflowInterrupt(void){

    MSB++; // increase extension variable with it the counter
    overflows every 2 147 483.648 seconds or almost 600 hours, around 25
    days so no need to worry about overflow

}


uint32_t micros(){

    uint32_t result;

    result=MSB<<16|TIM2->CNT;// add the counter extension to the
    counter itself and mulltiplyit by the time resolution to get the
    microseconds elapsed since initialization

    return result*500;

}


uint32_t millis(){

    return micros()/1000;// return millisecond passed since
    initialization

}


void delay(int ms){ // wait for some milliseconds without doing
    anything else

    uint32_t tini=millis();

    while(millis()-tini<ms);

}
```

## Annex 6.8 ADC configuratation

```
#include "adc.h"

#include "stm32f10x_adc.h"
```

```
#include "stdbool.h"

void BatteryLedInit() {
    //initialize the led pin that shows low battery warning
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOG, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOG, &GPIO_InitStructure);
}

void Adc_Init(void)
{
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_ADC1
, ENABLE ); // enable clock signals for the GPIO and the ADC

    RCC_ADCCLKConfig(RCC_PCLK2_Div6); //configure adc clock

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; //initialize pin 0 from
B port to be alternate input function
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    ADC_DeInit(ADC1);

    /* ADC1 configuration -----
-----*/
```

```
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;

    ADC_InitStructure.ADC_ScanConvMode = DISABLE;

    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;

    ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_None;

    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;

    ADC_InitStructure.ADC_NbrOfChannel = 1;

    ADC_Init(ADC1, &ADC_InitStructure);

    ADC_Cmd(ADC1, ENABLE);

    /* Enable ADC1 reset calibration register */
    ADC_ResetCalibration(ADC1);

    /* Check the end of ADC1 reset calibration register */
    while(ADC_GetResetCalibrationStatus(ADC1));

    /* Start ADC1 calibration */
    ADC_StartCalibration(ADC1);

    /* Check the end of ADC1 calibration */
    while(ADC_GetCalibrationStatus(ADC1));

    /* Start ADC1 Software Conversion */
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);

    BatteryLedInit();
}

u16 Get_Adc(u8 ch)
{
    ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_239Cycles5 );
    //start conversion on the channel requested
```

```
    /* Start ADC1 Software Conversion */

    ADC_SoftwareStartConvCmd(ADC1, ENABLE);

    //while(!(ADC1->SR&1<<1));

    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//wait until end of
conversion

    return ADC_GetConversionValue(ADC1);
}

void setLowBatLed(bool state){// set or reset the LED pin

    if(state)    GPIO_SetBits(GPIOG,GPIO_Pin_0);

    else GPIO_ResetBits(GPIOG,GPIO_Pin_0);

}

void checkBattery(void){//check the battery status

    u16 hexV;

    double volts;

    hexV=Get_Adc(ADC_Channel_8);//get adc value as 16 bit register

    volts=(double)3.3*((double)hexV)/(double)4095; //convert it to
volts using the formula from the appropriate section

    if(volts<1.8){

        setLowBatLed(true);//if voltage is lower than the threshold
set the pin

    }

    else setLowBatLed(false);//if not reset it in case it was set
before

}
```

## Annex 6.9 Timer 2 interrupt

```
*****

* @file    GPIO/IOToggle/stm32f10x_it.c
```



```
* @author   MCD Application Team

* @version  V3.5.0

* @date     08-April-2011

* @brief    Main Interrupt Service Routines.

*           This file provides template for all exceptions handler
and peripherals

*           interrupt service routine.

*****

/* Includes -----*/

#include "stm32f10x_it.h"

#include "Timer2.h"

#include "adc.h"

/*****
/*           STM32F10x Peripherals Interrupt Handlers          */
/*
/* Add here the Interrupt Handler for the used peripheral(s) (PPP),
for the */

/* available peripheral interrupt handler's name please refer to the
startup */

/* file (startup_stm32f10x_xx.s).
*/

*****/

void TIM2_IRQHandler(void){//Timer 2 interrupt. every 32 seconds

    checkBattery();

    overflowInterrupt();

    TIM2->SR&=~(1<<6); //remove interrupt flag

}
```

## Annex 6.10 LCD GUI

```
#include "lcd.h"
#include "string.h"
#include "font.h"
#include "delay.h"
#include "gui.h"

void GUI_DrawPoint(u16 x,u16 y,u16 color)//draws a single pixel update
window and colours it
{
    LCD_SetCursor(x,y);
    LCD_DrawPoint_16Bit(color);
}

void LCD_Fill(u16 sx,u16 sy,u16 ex,u16 ey,u16 color)
{
    u16 i,j;
    u16 width=ex-sx+1;
    u16 height=ey-sy+1;
    LCD_SetWindows(sx,sy,ex-1,ey-1);// creates an update window

    for(i=0;i<height;i++)
    {
        for(j=0;j<width;j++)
            LCD_WR_DATA(color); //fills all the update window with the
color in argument
    }
    LCD_SetWindows(0,0,lcddev.width-1,lcddev.height-1); //set update
window as the entire lcd
}

void LCD_Fill_Off(u16 sx,u16 sy,u16 ox,u16 oy,u16 color) //same as LCD
fill but using start+ width/height instead of start-end
{
    u16 ex=sx+ox;
    u16 ey=sy+oy;
    LCD_Fill(sx,sy,ex,ey,color);
}

void LCD_ShowChar(u16 x,u16 y,u16 fc, u16 bc, u8 num,u8 size,u8 mode)
{
    u8 temp;
    u8 pos,t;
    u16 colortemp=POINT_COLOR;

    num=num-' '; //to transform the ascii value into an index for the
font array
    LCD_SetWindows(x,y,x+size/2-1,y+size-1);//sets a window to fit the
character send
    if(!mode) // paints the background with bc colour
    {
        for(pos=0;pos<size;pos++)
        {
            temp=asc2_1608[num][pos]; //gets the pixel value from
the font table and prints the corresponding pixels to match the
defined size
        }
    }
}
```

```

        for(t=0;t<size/2;t++)
        {
            if(temp&0x01)LCD_DrawPoint_16Bit(fc);
            else LCD_DrawPoint_16Bit(bc);
            temp>>=1;
        }
    }
}
}else //same as above but without painting the background
{
    for(pos=0;pos<size;pos++)
    {
        temp=asc2_1608[num][pos];
        for(t=0;t<size/2;t++)
        {
            POINT_COLOR=fc;
            if(temp&0x01)LCD_DrawPoint(x+t,y+pos);
            temp>>=1;
        }
    }
}
POINT_COLOR=colortemp;
LCD_SetWindows(0,0,lcddev.width-1,lcddev.height-1);
}

void LCD_ShowString(u16 x,u16 y,u8 size,u8 *p,u8 mode)
{
    while((*p<='~')&&(*p>=' '))//print characters until one of them is
not in the font array
    {
        if(x>(lcddev.width-1)||y>(lcddev.height-1)) //dont print
passed the lcd borders
        return;
        LCD_ShowChar(x,y,POINT_COLOR,WHITE,*p,size,mode);
        x+=size/2;
        p++;
    }
}

u32 mypow(u8 m,u8 n)// m^n
{
    u32 result=1;
    while(n-->0)result*=m;
    return result;
}

void LCD_ShowNum(u16 x,u16 y,u32 num,u8 len,u8 size)//shows a number
padded with 0 o not depending on the arguments
{
    u8 t,temp;
    u8 enshow=0;
    for(t=0;t<len;t++)
    {
        temp=(num/mypow(10,len-t-1))%10;
        if(enshow==0&&t<(len-1))
        {
            if(temp==0)
            {

```

```
        LCD_ShowChar(x+(size/2)*t,y,POINT_COLOR,BACK_COLOR,'
',size,0);
        continue;
    }else enshow=1;

}

LCD_ShowChar(x+(size/2)*t,y,POINT_COLOR,BACK_COLOR,temp+'0',size,0);
}
}

void LCD_ShowStringCenter(u16 x, u16 y, u16 fc, u16 bc, u8 *str,u8
size,u8 mode)//same as show string but center it in the lcd
{
    u16 len=strlen((const char *)str);
    u16 x1=(lcddev.width-len*8)/2;
    LCD_ShowStringCenter(x+x1,y,fc,bc,str,size,mode);
}

void draw_sprite_uChar_scaled(u16 xStar,u16 yStar,u16 sprW,u16
sprH,const unsigned char *p,int scale){ // prints an array
representing a sprite, can be scaled to make it bigger

    int i,j,n=0;
    unsigned char picH,picL;
    LCD_SetWindows_Off(xStar,yStar,scale*sprW-1,scale*sprH-1);
    for(i=0;i<sprW*sprH;i++)
    {
        for(j=0;j<scale;j++){
            picL=(p+i*2); //the array is a u8 type, we need to
concatenate 2 bytes since pixels use 16 bits
            picH=(p+i*2+1);
            LCD_DrawPoint_16Bit(picH<<8|picL);
        }
        if((i+1)%sprW==0){//a row is complete
            if(n<scale-1){
                i=i-sprW;//print the same row again until it has
printed scale times
                n++;
            }else n=0;
        }
    }

    LCD_SetWindows(0,0,lcddev.width-1,lcddev.height-1);
}
```

## Annex 6.11 LCD.h

```
#ifndef __LCD_H
#define __LCD_H
#include "sys.h"
#include "stdlib.h"

/*****
*****
//=====0°¼$ÆÁÊÝ¼ÝÏß¼ÓÏß=====
=====//
STM32 PB×é¼Ó0°¼$ÆÁDB0~DB16,¼ÜÀÝÒÀ´ÎÎªDB0¼ÓPB0,..DB15¼ÓPB15.
//=====0°¼$ÆÁ¿ØÖËÏß¼ÓÏß=====
=====//
//LCD_CS      ¼ÓPC9 //Æ-Ñ;ÐÀ°Å
//LCD_RS      ¼ÓPC8 //¼Ä´æÆ÷/ÊÝ¼ÝÑ;ÕñÐÀ°Å
//LCD_WR      ¼ÓPC7 //Ð´ÐÀ°Å
//LCD_RD      ¼ÓPC6 //¶ÁÐÀ°Å
//LCD_RST     ¼ÓPC5 //´Í»ÐÀ°Å
//LCD_LED     ¼ÓPC10 //±³â¿ØÖÆÐÀ°Å(,ßµÇÆ¼µääÁ)
//=====¼ÄpÆÁ´Ý¼ÓÏß=====
=====//
//²»Ê¹ÓÄ´ÝÄp»ðÕßÄ£¿é±¼Ê¹²»´ø´ÝÄp£-Ôò¿Ê²»Á-¼Ó
//MO(MISO)    ¼ÓPC2 //SPI×ÜÏßÊä³ö
//MI(MOSI)    ¼ÓPC3 //SPI×ÜÏßÊäÊë
//PEN         ¼ÓPC1 //´ÝÄpÆÁÔÐ¶ÍÐÀ°Å
//TCS         ¼ÓPC13 //´ÝÄpICE-Ñ;
//CLK         ¼ÓPC0 //SPI×ÜÏßÊ±ÔÖ
*****
*****/

//LCD000ª²ÎÊÝ¼
typedef struct
{
    u16 width;           //LCD ¿í¶È
    u16 height;          //LCD ,ß¶È
    u16 id;               //LCD ID
    u8 dir;               //°áÆÁ»¹ÊÇÊúÆÁ¿ØÖË£°0£-ÊúÆÁ£»1£-°áÆÁ¿;£
    u16 wramcmd;          //¿ªÊ¼Ð´gramÖ,Áî
    u16 setxcmd;          //ÊèÖÄx×ø±èÖ,Áî
    u16 setycmd;          //ÊèÖÄy×ø±èÖ,Áî
} _lcd_dev;

//LCD²ÎÊÝ
extern _lcd_dev lcddev; //¹ÜÀíLCD000ª²ÎÊÝ
////////////////////////////////////////ÓÄ»$ÄäÖÄÇø////////////////////////////////////////
////////////////////////////////////////
//Ö$³ÖªáÊúÆÁ¿îÊÜ¶´´ÒäÇÐ»»£-Ö$³Ö8/16Î»Ä£Ê¼ÇÐ»»
#define USE_HORIZONTAL      1 //¶´´ÒäÊÇ·ñÊ¹ÓÄªáÆÁ
0,²»Ê¹ÓÄ.1,Ê¹ÓÄ.
#define LCD_USE8BIT_MODEL   0 //¶´´ÒäÊÝ¼Ý×ÜÏßÊÇ·ñÊ¹ÓÄ8Î»Ä£Ê¼
0,Ê¹ÓÄ16Î»Ä£Ê¼.1,Ê¹ÓÄ8Î»Ä£Ê¼
////////////////////////////////////////
////////////////////////////////////////
//¶´´ÒäLCDµÄ³ß´Ç
#if USE_HORIZONTAL==1 //Ê¹ÓÄªáÆÁ
#define LCD_W 480
#define LCD_H 320
#else
#define LCD_W 320
#define LCD_H 480
#endif
#endif
```

```
//TFTLCD²:·Öíãðªµ÷óÃµÃ°¯Êý
extern u16 POINT_COLOR; //Ä-Èï°iÉ«
extern u16 BACK_COLOR; //±³¾°ÑÕÉ«.Ä-Èïïª°×É«

////////////////////////////////////
//-----LCD¶È¿Ú¶¯Òã-----
//QDtechÈ«ïµÁÐÄ£:é²ÉÓÄÄÈÊý¼«¹Û¿ØÖÆ±³¹ääÄÄð£-ÓÄ»§Ò²¿ÉÒÖ¼ÓPWMµ÷¼Ú±³¹ääÄ¶
È
#define LCD_LED PCout(10) //LCD±³¹ää PC10
//È¿¹ûÈ¹ÓÄ¹Û·¼¿ã°¯Êý¶¯ÒãïÃÄÐµ×²ää-ÈÛ¶È¼«»ääïÃµµ¼14Ö;Ä¿Äê£-¼²¯Òé²ÉÓÄïÒÈ¼
ïÆ¼ð·¼·
//ÒÖïÃïÖ¶¯ÒãÖ±¼Ö²Û×÷¼Ä´æÆ÷£-¿iÈÛïÖ²Û×÷£-È¿ÆÄÈÛÄÊ¿ÉÒÖ´ïµ¼28Ö;Ä¿Äê£;

#define LCD_RS_SET GPIOA->BSRR=1<<0 //Êý³¼Ý/ÃüÁî PC8
#define LCD_WR_SET GPIOA->BSRR=1<<9 //Ð´Êý³¼Ý PC7
#define LCD_CS_SET GPIOA->BSRR=1<<2 //Æ-Ñ;¶È¿Ú PC9
#define LCD_RST_SET GPIOA->BSRR=1<<3 //´ï» PC5
#define LCD_RD_SET GPIOA->BSRR=1<<1 //¶ÄÊý³¼Ý PC6

#define LCD_RS_CLR GPIOA->BRR=1<<0 //Êý³¼Ý/ÃüÁî PC8
#define LCD_WR_CLR GPIOA->BRR=1<<9 //Ð´Êý³¼Ý PC7
#define LCD_CS_CLR GPIOA->BRR=1<<2 //Æ-Ñ;¶È¿Ú PC9
#define LCD_RST_CLR GPIOA->BRR=1<<3 //´ï» PC5
#define LCD_RD_CLR GPIOA->BRR=1<<1 //¶ÄÊý³¼Ý PC6
//PB0~15,×÷ïªÊý³¼ïß
//×¿Òáf°È¿¹ûÈ¹ÓÄ8î»Ä£Ê¼Êý³¼×Ûïß£-Òð°°§ÆÄµÄÊý³¼,ß8î»Ê¿¼Óµ¼MCUµÄ,ß8î»×Û
ïßÉï
//¼ÜÄý£°È¿¹û¼Ö8î»Ä£Ê¼Òð±¼Ê¼Äý¼Öïßïª°°°§ÆÄDB10-
DB17¶ÖÖ¼¼ÖÖÄµ¥Æ-»úGPIOB_Pin8-GPIOB_Pin15
//¼ÜÄý£°È¿¹ûÊ¿16î»Ä£Ê¼£°DB0-DB7·Ö÷ð¼ÖGPIOB_Pin0-GPIOB_Pin7,DB10-
DB17¶ÖÖ¼¼ÖÖÄµ¥Æ-»úGPIOB_Pin8-GPIOB_Pin15
#define DATAOUT(x) GPIOC->ODR=x; //Êý³¼ÝÊä³ð
#define DATAIN GPIOC->IDR; //Êý³¼ÝÊäÈè
////////////////////////////////////

//É·Äè·¼ïÒ¶¯Òã
#define L2R_U2D 0 //´ó×óµ¼ÖÖ,´ÓÉïµ¼ïÃ
#define L2R_D2U 1 //´ó×óµ¼ÖÖ,´ÓïÃµ¼Èï
#define R2L_U2D 2 //´ÓÖðµ¼×ó,´ÓÉïµ¼ïÃ
#define R2L_D2U 3 //´ÓÖðµ¼×ó,´ÓïÃµ¼Èï

#define U2D_L2R 4 //´ÓÉïµ¼ïÃ,´ó×óµ¼ÖÖ
#define U2D_R2L 5 //´ÓÉïµ¼ïÃ,´ÓÖðµ¼×ó
#define D2U_L2R 6 //´ÓïÃµ¼Èï,´ó×óµ¼ÖÖ
#define D2U_R2L 7 //´ÓïÃµ¼Èï,´ÓÖðµ¼×ó

#define DFT_SCAN_DIR L2R_U2D //Ä-ÈïµÄÉ·Äè·¼ïÒ

//»±ÊÑÕÉ«
#define WHITE 0xFFFF
#define BLACK 0x0000
#define BLUE 0x001F
#define BRED 0XF81F
#define GRED 0XFFE0
#define GBLUE 0X07FF
#define RED 0xF800
#define MAGENTA 0xF81F
#define GREEN 0x07E0
```

```
#define CYAN          0x7FFF
#define YELLOW        0xFFE0
#define BROWN         0xBC40 //×ØÉ«
#define BRRED         0xFC07 //×Ø°iÉ«
#define GRAY          0x8430 //»ØÉ«
//GUIÑÖÉ«

#define DARKBLUE       0x01CF //ÉiÀ¶É«
#define LIGHTBLUE      0x7D7C //Ç³À¶É«
#define GRAYBLUE       0x5458 //»ØÀ¶É«
//ÖÖÉiÈÿÉ«îªPANELµÄÑÖÉ«

#define LIGHTGREEN      0x841F //Ç³ÂiÉ«
//#define LIGHTGRAY    0xEF5B //Ç³»ØÉ« (PANNEL)
#define LGRAY          0xC618 //Ç³»ØÉ« (PANNEL), ´°îâ±³¼°É«

#define LGRAYBLUE      0xA651 //Ç³»ØÀ¶É« (ÖÐªä² äÑÖÉ«)
#define LBBLUE         0x2B12 //Ç³×ØÀ¶É« (Ñ;ÖñîöÄ;µÄ·´É«)

extern ul6 BACK_COLOR, POINT_COLOR ;

void LCD_Init(void);
void LCD_DisplayOn(void);
void LCD_DisplayOff(void);
void LCD_Clear(ul6 Color);
void LCD_SetCursor(ul6 Xpos, ul6 Ypos);
void LCD_DrawPoint(ul6 x,ul6 y);//»µã
ul6 LCD_ReadPoint(ul6 x,ul6 y); //¶Áµã
void LCD_DrawLine(ul6 x1, ul6 y1, ul6 x2, ul6 y2);
void LCD_DrawRectangle(ul6 x1, ul6 y1, ul6 x2, ul6 y2);
void LCD_SetWindows(ul6 xStar, ul6 yStar,ul6 xEnd,ul6 yEnd);
void LCD_DrawPoint_16Bit(ul6 color);
ul6 LCD_RD_DATA(void);//¶ÁÉ;LCDËÿ¼ÿ
void LCD_WriteReg(u8 LCD_Reg, ul6 LCD_RegValue);
void LCD_WR_DATA(ul6 data);
ul6 LCD_ReadReg(u8 LCD_Reg);
void LCD_WriteRAM_Prepare(void);
void LCD_WriteRAM(ul6 RGB_Code);
ul6 LCD_ReadRAM(void);
ul6 LCD_BGR2RGB(ul6 c);
void LCD_SetParam(void);
void LCD_SetWindows_Off(ul6 xStar, ul6 yStar,ul6 xOff,ul6 yOff);

//Ëç¹ûÈÖË»³õµÄËÛ¶²»¹»¿î£¬¿ÖÖË¹ÖÄiÄÄæµÄ°ê¶²Òä,îá,ßËÛ¶².
//×çÒäÒªËµðlcd.cÖÐvoid LCD_WR_DATA(ul6 data)°ËËÿ¶²Òä¶²
/*
#if LCD_USE8BIT_MODEL==1//Ë¹ÖÄ8î»²çÐÐËÿ¼ÿ×ÛißÄ£Ë²
    #define LCD_WR_DATA(data){\
        LCD_RS_SET;\
        LCD_CS_CLR;\
        DATAOUT(data);\
        LCD_WR_CLR;\
        LCD_WR_SET;\
        DATAOUT(data<<8);\
        LCD_WR_CLR;\
        LCD_WR_SET;\
        LCD_CS_SET;\
    }
#else//Ë¹ÖÄ16î»²çÐÐËÿ¼ÿ×ÛißÄ£Ë²
    #define LCD_WR_DATA(data){\
        LCD_RS_SET;\

```

```
        LCD_CS_CLR;\
        DATAOUT(data);\
        LCD_WR_CLR;\
        LCD_WR_SET;\
        LCD_CS_SET;\
    }
#endif
*/

//9320/9325 LCD
#define R0          0x00
#define R1          0x01
#define R2          0x02
#define R3          0x03
#define R4          0x04
#define R5          0x05
#define R6          0x06
#define R7          0x07
#define R8          0x08
#define R9          0x09
#define R10         0x0A
#define R12         0x0C
#define R13         0x0D
#define R14         0x0E
#define R15         0x0F
#define R16         0x10
#define R17         0x11
#define R18         0x12
#define R19         0x13
#define R20         0x14
#define R21         0x15
#define R22         0x16
#define R23         0x17
#define R24         0x18
#define R25         0x19
#define R26         0x1A
#define R27         0x1B
#define R28         0x1C
#define R29         0x1D
#define R30         0x1E
#define R31         0x1F
#define R32         0x20
#define R33         0x21
#define R34         0x22
#define R36         0x24
#define R37         0x25
#define R40         0x28
#define R41         0x29
#define R43         0x2B
#define R45         0x2D
#define R48         0x30
#define R49         0x31
#define R50         0x32
#define R51         0x33
#define R52         0x34
#define R53         0x35
#define R54         0x36
#define R55         0x37
#define R56         0x38
#define R57         0x39
#define R59         0x3B
```



```
#define R60          0x3C
#define R61          0x3D
#define R62          0x3E
#define R63          0x3F
#define R64          0x40
#define R65          0x41
#define R66          0x42
#define R67          0x43
#define R68          0x44
#define R69          0x45
#define R70          0x46
#define R71          0x47
#define R72          0x48
#define R73          0x49
#define R74          0x4A
#define R75          0x4B
#define R76          0x4C
#define R77          0x4D
#define R78          0x4E
#define R79          0x4F
#define R80          0x50
#define R81          0x51
#define R82          0x52
#define R83          0x53
#define R96          0x60
#define R97          0x61
#define R106         0x6A
#define R118         0x76
#define R128         0x80
#define R129         0x81
#define R130         0x82
#define R131         0x83
#define R132         0x84
#define R133         0x85
#define R134         0x86
#define R135         0x87
#define R136         0x88
#define R137         0x89
#define R139         0x8B
#define R140         0x8C
#define R141         0x8D
#define R143         0x8F
#define R144         0x90
#define R145         0x91
#define R146         0x92
#define R147         0x93
#define R148         0x94
#define R149         0x95
#define R150         0x96
#define R151         0x97
#define R152         0x98
#define R153         0x99
#define R154         0x9A
#define R157         0x9D
#define R192         0xC0
#define R193         0xC1
#define R229         0xE5
#endif
```

## Annex 6.12 LCD.c

```
#include "lcd.h"
#include "stdlib.h"
#include "usart.h"
#include "delay.h"

////////////////////////////////////
////////////////////////////////////

_lcd_dev lcddev;
u16 POINT_COLOR = 0xFFFF, BACK_COLOR = 0x0000;
u16 DeviceCode;

//*****
//*****
void LCD_WR_REG(u8 data)
{
    LCD_RS_CLR; //D'µÖ·
    LCD_CS_CLR;
    LCD_WR_CLR;
    DATAOUT(data);
    LCD_WR_SET;
    LCD_CS_SET;
}

//*****
//*****
void LCD_WR_DATA(u16 data)
{
    LCD_RS_SET;
    LCD_CS_CLR;

    LCD_WR_CLR;
    DATAOUT(data);
    LCD_WR_SET;
    LCD_CS_SET;
}

//*****
//*****
void LCD_DrawPoint_16Bit(u16 color)
{
    LCD_WR_DATA(color);
}

//*****
//*****
void LCD_WriteReg(u8 LCD_Reg, u16 LCD_RegValue)
{
    LCD_WR_REG(LCD_Reg);
    LCD_WR_DATA(LCD_RegValue);
}

//*****
//*****
void LCD_WriteRAM_Prepare(void)
{
    LCD_WR_REG(lcddev.wramcmd);
}
```

```
//*****
//*****
void LCD_DrawPoint(u16 x,u16 y)
{
    LCD_SetCursor(x,y); //ÉèÖÃ¹â±êÎ»ÖÃ
#ifdef LCD_USE8BIT_MODEL==1
    LCD_CS_CLR;
    LCD_RD_SET;
    LCD_RS_SET; //Ð´µØÖ·
    DATAOUT(POINT_COLOR>>8);
    LCD_WR_CLR;
    LCD_WR_SET;
    DATAOUT(POINT_COLOR);
    LCD_WR_CLR;
    LCD_WR_SET;
    LCD_CS_SET;
#else
    LCD_WR_DATA(POINT_COLOR);
#endif
}

//*****
//*****
void LCD_Clear(u16 Color)
{
    u32 index=0;
    u32 total=0;
    total=lcddev.width*lcddev.height;
    LCD_SetWindows(0,0,lcddev.width-1,lcddev.height-1);
    LCD_RS_SET;
    LCD_CS_CLR;

    for(index=0;index<total;index++)
    {
        //LCD_WR_DATA(Color);
        DATAOUT(Color);
        LCD_WR_CLR;
        LCD_WR_SET;
    }
}

//*****
//*****
void LCD_GPIOInit(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOA|RCC_APB2Periph_AFIO, ENABLE);
    GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable , ENABLE);

    GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_9; //GPIO_Pin_10
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //íÆíîÊä³ö
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure); //GPIOC
```

```
GPIO_SetBits(GPIOA,GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pi  
n_4|GPIO_Pin_5|GPIO_Pin_9);
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All; //  
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //íÆíîÊä³ö  
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;  
    GPIO_Init(GPIOC, &GPIO_InitStructure); //GPIOA  
    GPIO_SetBits(GPIOC,GPIO_Pin_All);  
}
```

```
//*****  
//*****
```

```
void LCD_RESET(void)  
{  
    LCD_RST_CLR;  
    delay_ms(100);  
    LCD_RST_SET;  
    delay_ms(50);  
}
```

```
//*****  
//*****
```

```
void LCD_Init(void)  
{
```

```
    LCD_GPIOInit();  
    LCD_RESET();
```

```
LCD_WR_REG(0xB4); //Set RM, DM  
LCD_WR_DATA(0x00); //
```

```
LCD_WR_REG(0xC0); //Set Panel Driving  
LCD_WR_DATA(0x10); //REV SM GS  
LCD_WR_DATA(0x3B); // NL[5:0]  
LCD_WR_DATA(0x00); //SCN[6:0]  
LCD_WR_DATA(0x02); //NDL 0 PTS[2:0]  
LCD_WR_DATA(0x11); //PTG ISC[3:0]
```

```
LCD_WR_REG(0xC1); //  
LCD_WR_DATA(0x10); //line inversion
```

```
LCD_WR_REG(0xC8); //Set Gamma  
LCD_WR_DATA(0x00); //KP1,KP0  
LCD_WR_DATA(0x46); //KP3,KP2  
LCD_WR_DATA(0x12); //KP5,KP4  
LCD_WR_DATA(0x20); //RP1,RP0  
LCD_WR_DATA(0x0c); //VRP0 01  
LCD_WR_DATA(0x00); //VRP1  
LCD_WR_DATA(0x56); //KN1,KN0  
LCD_WR_DATA(0x12); //KN3,KN2  
LCD_WR_DATA(0x67); //KN5,KN4  
LCD_WR_DATA(0x02); //RN1,RN0  
LCD_WR_DATA(0x00); //VRN0  
LCD_WR_DATA(0x0c); //VRN1 01
```

```
LCD_WR_REG(0xD0); //Set Power  
LCD_WR_DATA(0x44); //DDVDH :5.28  
LCD_WR_DATA(0x42); // BT VGH:15.84 VGL:-7.92  
LCD_WR_DATA(0x06); //VREG1 4.625V
```

```
LCD_WR_REG(0xD1); //Set VCOM
LCD_WR_DATA(0x73); //VCOMH
LCD_WR_DATA(0x16);

LCD_WR_REG(0xD2);
LCD_WR_DATA(0x04);
LCD_WR_DATA(0x22); //12

LCD_WR_REG(0xD3);
LCD_WR_DATA(0x04);
LCD_WR_DATA(0x12);

LCD_WR_REG(0xD4);
LCD_WR_DATA(0x07);
LCD_WR_DATA(0x12);

LCD_WR_REG(0xE9); //Set Panel
LCD_WR_DATA(0x00);

LCD_WR_REG(0xC5); //Set Frame rate
LCD_WR_DATA(0x08); //61.51Hz

LCD_WR_REG(0X36);
LCD_WR_DATA(0X0a);

LCD_WR_REG(0X3A);
LCD_WR_DATA(0X55);

LCD_WR_REG(0x11); //Sleep Out
delay_ms(120);

LCD_WR_REG(0x29); //Display On
delay_ms(5);

LCD_SetParam(); //ÉèÖÃLCD²îÊý
LCD_LED=1; //µãÁÁ±³â
LCD_Clear(WHITE);
}
//*****
//*****

void LCD_SetWindows(u16 xStar, u16 yStar,u16 xEnd,u16 yEnd)
{

    LCD_WR_REG(lcddev.setxcmd);
    LCD_WR_DATA(xStar>>8);
    LCD_WR_DATA(xStar);
    LCD_WR_DATA(xEnd>>8);
    LCD_WR_DATA(xEnd);

    LCD_WR_REG(lcddev.setycmd);
    LCD_WR_DATA(yStar>>8);
    LCD_WR_DATA(yStar);
    LCD_WR_DATA(yEnd>>8);
    LCD_WR_DATA(yEnd);

    LCD_WriteRAM_Prepare();
}
```

```
//*****
//*****

void LCD_SetWindows_Off(u16 xStar, u16 yStar,u16 xOff,u16 yOff)
{
    u16 xEnd=xStar+xOff;
    u16 yEnd=yStar+yOff;

    LCD_WR_REG(lcddev.setxcmd);
    LCD_WR_DATA(xStar>>8);
    LCD_WR_DATA(xStar);
    LCD_WR_DATA(xEnd>>8);
    LCD_WR_DATA(xEnd);

    LCD_WR_REG(lcddev.setycmd);
    LCD_WR_DATA(yStar>>8);
    LCD_WR_DATA(yStar);
    LCD_WR_DATA(yEnd>>8);
    LCD_WR_DATA(yEnd);

    LCD_WriteRAM_Prepare();

}

//*****
//*****

void LCD_SetCursor(u16 Xpos, u16 Ypos)
{
    LCD_SetWindows(Xpos,Ypos,Xpos+1,Ypos+1);
}

//*****
//*****

void LCD_SetParam(void)
{
    lcddev.setxcmd=0x2A;
    lcddev.setycmd=0x2B;
    lcddev.wramcmd=0x2C;
    #if USE_HORIZONTAL==1 //Horizontal
        lcddev.dir=1;
        lcddev.width=480;
        lcddev.height=320;
        LCD_WriteReg(0x36,0x3B); //BGR==1,MY==1,MX==0,MV==1 63
    #else //Vertical
        lcddev.dir=0;
        lcddev.width=320;
        lcddev.height=480;
        LCD_WriteReg(0x36,0x0A); //BGR==1,MY==0,MX==0,MV==0
    #endif
}
```

## Annex 6.13 SPI.c

```
#include "spi.h"

SPI_InitTypeDef SPI_InitStructure;

void SPIx_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA|RCC_APB2Periph_SPI1,
    ENABLE );

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_6 |
    GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // 'ÓÃÍÆíîÊä³ò
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_SetBits(GPIOA,GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7);

    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    // ÊèÖÄSPIµ¥íò»ðÖßÊ«íòµÄÊý³ÝÄ£Ê½:SPIÊèÖÄí²Ê«íòÊ«Ê«¹²
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
    // ÊèÖÄSPI¹²×÷Ä£Ê½:ÊèÖÄí²Ö÷SPI
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
    // ÊèÖÄSPIµÄÊý³Ý´ð;SPI·Êí½ÖÊÖ8í»ð;½á¹¹
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
    // Ñ;ÔñÄÊ´ðÊ±ÖóµÄíÊí-:Ê±ÖóÐù;Öß
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
    // Êý³Ý²¶ñóÓµÚ¶,öÊ±ÖóNØ
    SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
    // NSSÐ°ÄÊÖ²¶±NSS¹½Ä£»¹ÊÇÊí¶±¹Ê¹ÖÄSSIí£©¹ÜÁ:ÄÜ²;NSSÐ°ÄÊÖSSIí
    ;ØÖÊ
    SPI_InitStructure.SPI_BaudRatePrescaler =
    SPI_BaudRatePrescaler_256;
    // ¶¹²¹ìØÄÊÖ·ÖµµÄö:²¹ìØÄÊÖ·Öµöµí²²56
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
    // Ö,¶¹Êý³Ý«Êä´óMSBí»¹ÊÇLSBí;¹Ê:Êý³Ý«Êä´óMSBí;¹Ê
    SPI_InitStructure.SPI_CRCPolynomial = 7; //CRCÖµ¹ÆÊäµÄ¶äíî½
    SPI_Init(SPI1, &SPI_InitStructure);
    // ù³ÝSPI_InitStructÖÐ,¶¹²¹Êý³öÊ¹¹ÊÊèSPIx¹²Æ÷

    SPI_Cmd(SPI1, ENABLE); // Ê¹ÄÜSPIíäÊè

    SPIx_ReadWriteByte(0xff); // Êð¶¹«Êä
}

// SPI ÊÜÊèÖÄ³¹Êý
// SpeedSet:
// SPI_BaudRatePrescaler_2 2·Öµ (SPI 36M@sys 72M)
// SPI_BaudRatePrescaler_8 8·Öµ (SPI 9M@sys 72M)
// SPI_BaudRatePrescaler_16 16·Öµ (SPI 4.5M@sys 72M)
// SPI_BaudRatePrescaler_256 256·Öµ (SPI 281.25K@sys 72M)

void SPIx_SetSpeed(u8 SpeedSet)
{
    SPI_InitStructure.SPI_BaudRatePrescaler = SpeedSet ;
    SPI_Init(SPI1, &SPI_InitStructure);
    SPI_Cmd(SPI1,ENABLE);
}
```

```
}

u8 SPIx_ReadWriteByte(u8 TxData)
{
    u8 retry=0;

    while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET)
    {
        retry++;
        if(retry>200)return 0;
    }

    SPI_I2S_SendData(SPI1, TxData);
    retry=0;

    while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET)
    {
        retry++;
        if(retry>200)return 0;
    }

    return SPI_I2S_ReceiveData(SPI1);
}
```



## Annex 6.14 Sprite Arrays

```
const unsigned char gImage_alien1a[136] = {
0X00,0X10,0X08,0X00,0X08,0X00,0X01,0X1B,
0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0X00,0X00,
0X00,0X00,
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0X00,0X00,
0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,
0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0X00,0X00,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,
0X00,0X00,
0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0XFF,0XFF,
0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,
0X00,0X00,};

const unsigned char gImage_alien1b[136] = {
0X00,0X10,0X08,0X00,0X08,0X00,0X01,0X1B,
0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0X00,0X00,
0X00,0X00,
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0X00,0X00,
0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,
0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,
0X00,0X00,
0X00,0X00,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,
0X00,0X00,
0XFF,0XFF,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,
0XFF,0XFF
};

const unsigned char gImage_alien2a[184] = {
0X00,0X10,0X0B,0X00,0X08,0X00,0X01,0X1B,
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,
0X00,0X00,
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0X00,0X00,
0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,
0X00,0X00,};
```

```
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,  
0X00,0X00  
};
```

```
const unsigned char gImage_alien2b[184] = {  
0X00,0X10,0X0B,0X00,0X08,0X00,0X01,0X1B,  
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,  
0X00,0X00,  
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,  
0X00,0X00,  
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,  
0XFF,0XFF,  
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XDF,0XFF,0XFF,0XFF,  
0XFF,0XFF,  
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XDF,0XFF,0XFF,0XFF,  
0X00,0X00,  
0XFF,0XFF,0XDF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,  
0XDF,0XFF,  
0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,  
0X00,0X00,  
};
```

```
const unsigned char gImage_alien3a[200] = {  
0X00,0X10,0X0C,0X00,0X08,0X00,0X01,0X1B,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,  
0XFF,0XFF,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,  
0XFF,0XFF,  
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,  
0X00,0X00,  
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,  
0XFF,0XFF,  
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,  
0X00,0X00,  
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,  
0XFF,0XFF,  
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,  
0XFF,0XFF,  
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,  
0XFF,0XFF,  
0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0XFF,0XFF,  
0X00,0X00,  
0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,  
0XFF,0XFF,  
};
```

```
const unsigned char gImage_alien3b[200] = {
0X00,0X10,0X0C,0X00,0X08,0X00,0X01,0X1B,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0X00,0X00,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0X00,0X00,
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XDF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,
0XFF,0XFF,0XDF,0XFF,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,
0XFF,0XFF,
0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0X00,0X00,
0X00,0X00,
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,
0X00,0X00,
0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,
0XFF,0XFF,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0XFF,0XFF,0X00,0X00,
0X00,0X00,
};
```

```
const unsigned char gImage_alienDead[216] = {
0X00,0X10,0X0D,0X00,0X08,0X00,0X01,0X1B,
0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,
0X00,0X00,
0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,
0XFF,0XFF,
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,
0XFF,0XFF,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,
0XFF,0XFF,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0XFF,0XFF,0X00,0X00,
0X00,0X00,
0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,
0XFF,0XFF,
0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,
0X00,0X00,
};
```

```
const unsigned char gImage_player[248] = {
0X00,0X10,0X0F,0X00,0X08,0X00,0X01,0X1B,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0XE0,0X07,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,
0XE0,0X07,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0XE0,0X07,
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,
0XE0,0X07,
0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,
0XE0,0X07,
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,
0XE0,0X07,
0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,
0XE0,0X07,
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,
0XE0,0X07,
0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,
0XE0,0X07,
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,
0X00,0X00,
};
```

```
const unsigned char gImage_playerDead[248] = {
0X00,0X10,0X0F,0X00,0X08,0X00,0X01,0X1B,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,0XE0,0X07,0X00,0X00,
0XE0,0X07,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0XE0,0X07,
0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,
0X00,0X00,
0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0X00,0X00,0XE0,0X07,0XE0,0X07,
0X00,0X00,
0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,
0XE0,0X07,
0X00,0X00,0XE0,0X07,0XE0,0X07,0X00,0X00,0XE0,0X07,0X00,0X00,0XE0,0X07,
0X00,0X00,
```

[illegible]

```
0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,  
0X00,0X00,  
0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,0X00,0X00,0XE0,0X07,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0XE0,0X07,0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,0XE0,0X07,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,  
0X00,0X00,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,  
0X00,0X00,  
0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,};
```

```
const unsigned char gImage_shield40[668] = {  
0X00,0X10,0X16,0X00,0X0F,0X00,0X01,0X1B,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0XE0,0X07,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,  
0X00,0X00,
```

0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0X00,0X00,  
0X00,0X00,  
0XE0,0X07,0X00,0X00,0XE0,0X07,0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0X00,0X00,0XE0,0X07,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0X00,0X00,  
0XE0,0X07,0X00,0X00,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,

[illegible]



[illegible]

```
const unsigned char gImage_shield100[668] = {  
    0x00, 0x10, 0x16, 0x00, 0x0F, 0x00, 0x01, 0x1B,  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x07, 0xE0, 0x07,  
    0xE0, 0x07,
```

155

```
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE0,0X07,0XE0,0X07,0XE0,0X07,  
0XE0,0X07,  
0XE0,0X07,0XE0,0X07,};
```

```
const unsigned char gImage_shot[56] = {  
0X00,0X10,0X03,0X00,0X08,0X00,0X01,0X1B,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,  
0XFF,0XFF,  
0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,  
0X00,0X00,  
0XFF,0XFF,0X00,0X00,0X00,0X00,0XFF,0XFF,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
};
```

```
const unsigned char gImage_UFO[232] = {  
0X00,0X10,0X10,0X00,0X07,0X00,0X01,0X1B,  
0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0XE4,0XE8,0XE4,0XE8,  
0XE4,0XE8,  
0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0X00,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,  
0XE4,0XE8,  
0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0X00,0X00,0X00,0X00,  
0X00,0X00,  
0X00,0X00,0X00,0X00,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,  
0XE4,0XE8,  
0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,  
0XE4,0XE8,  
0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,  
0XE4,0XE8,  
0X00,0X00,0X00,0X00,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0X00,0X00,0X00,0X00,  
0XE4,0XE8,  
0XE4,0XE8,0X00,0X00,0X00,0X00,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0X00,0X00,  
0X00,0X00,  
0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,  
0XE4,0XE8,  
0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,  
0XE4,0XE8,  
0X00,0X00,0X00,0X00,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0X00,0X00,0X00,0X00,  
0XE4,0XE8,  
0XE4,0XE8,0X00,0X00,0X00,0X00,0XE4,0XE8,0XE4,0XE8,0XE4,0XE8,0X00,0X00,  
0X00,0X00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0xE4,0xE8,0x00,0x00,0x00,0x00,0x00,0x00,  
0x00,0x00,  
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xE4,0xE8,0x00,0x00,0x00,0x00,  
0x00,0x00,  
};
```